UNIVERSITY OF OSLO

Department of Informatics

System for face detection on a mobile phone using Java technology

Master Thesis

Nancy Flores-Cuautle
November 2007

"After climbing a great hill,
one only finds that there are many more hills to climb"

Nelson Mandela

Acknowledge

This report describes my work undertaken for the Masters degree in Microelectronic Systems at the Department of Informatics, in the program for Electronics and Computer Technology at the University of Oslo. The thesis work was carried from January 2007 to November 2007.

I would like to express my deep and sincere gratitude to my research advisors Prof. Jim Tørresen and Associate Prof. Mats Høvin for their patience and helpful advice.

My loving thanks to my mother, father and brother. Without their encouragement and understanding, it would have been impossible for me to finish this work. And finally my special gratitude goes to my friends and family for their support trough this journey.

Nancy Flores Cuautle

Oslo, November 2007

Abstract

For a robot to interact with humans around its environment is necessary to achieve a communication that seems as natural as possible. Humans when talking to each other, look directly into their faces to indicate attention. A robot, on the other hand has to mimic this interaction by turning its point of attention towards the face of the human.

In this project a system for face detection on a mobile phone is presented. The sensor used for the input of the image is a mobile phone camera. The reason to use a mobile phone is the price and accessibility of the equipment. In addition, a mobile phone has integrated technologies such as Bluetooth and Java programming environment that are useful for the development of this project.

A research on the existing face detection algorithms is done. The bases for Java programming and description of the technologies to use are presented. Moreover, the implementation is proposed with three approaches: *Detection of Dark Areas, Motion analysis* and *Skin color detection*. Finally, the results are compared to various situations of luminance and position. With these implementations the algorithm can find faces with an accuracy rate of 87%.

The analysis of the image generates a coordinate (x, y) of the face location in the image. This coordinate may be sent to a robot and this turn towards the face. This results in an intuitive way of interaction between humans and robots. In other words, I am creating "the eyes" of a robot for human-robot interaction.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABS-plastic	-	Acrylonitrile Butadiene Styrene- Common thermoplastic
AC	-	Alternating Current
ACM	-	Association for Computing Machinery
AI	-	Artificial Intelligence
AIBO	-	Artificial Intelligence roBOt, Sony
AMS	-	Application Management Software
API	-	Application Programming Interfaces
B	-	Blue component in RGB color space
B′	-	Blue component in normalized RGB color space
BISM	-	Bluetooth Intelligent Serial Modules
Blu2	-	EZURiO Development kit. Specifications: Appendix A
Cb	-	Blue component of chrominance in Y′CbCr color space
CDN	-	Connected Device Configuration
CLDC	-	Connected Limited Device Configuration
Cr	-	Red component of chrominance in Y′CbCr color space
DRAM	-	Dynamic Random Access Memory
EDR	-	Enhanced Data Rate
ELFA	-	Electronics Supplier of Northern Europe
ETA	-	Electronic Travel Aids
G	-	Green component in RGB color space
G′	-	Green component in normalized RGB color space
GA	-	Genetic Algorithm
GCF	-	Generic Connection Framework
GHz	-	Giga Hertz
GPRS	-	General Packet Radio Service
GSM	-	Global System for Mobile communication (originally from Groupe Spécial Mobile)
GUI	-	Graphical User Interface
HCI	-	Human-Computer Interaction
HRI	-	Human-Robot Interaction
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	Secure Hypertext Transfer Protocol
IBM	-	International Business Machines Corporation
IDE	-	Integrate Development Environments
IDE	-	Java Integrated Development Environments

IEEE	-	Institute of Electrical and Electronics Engineers
J2EE	-	Java 2 Enterprise Edition
J2ME	-	Java 2 Micro Edition
J2SE	-	Java 2 Standard Edition
JAD	-	Java Application Descriptor
JAR	-	Java ARchive
JCP	-	Java Community Process
JNI	-	Java Native Interface
JSR	-	Java Specification Request
JTWI	-	Java Technology for the Wireless Industry
JVM	-	Java Virtual Machine
KVM	-	K Virtual Machine
L2CAP	-	Logical Link Control and Adaption Protocol
MB	-	Megabyte
Mbps	-	Mega byte per second
ME	-	Micro Edition
MID	-	Mobile Information Devices
MIDlet	-	Java program for embedded devices to use J2ME
MIDP	-	Mobile Information Device Profile
MVC	-	Model-View-Controller
OEM	-	Original Equipment Manufacturer
OS	-	Operating System
OTA	-	Over The Air
PC	-	Personal Computer
PCA	-	Principal Component Analysis
PDA	-	Personal Digital Assistants
PKI	-	Public Key Infrastructure
PM	-	Permanent Magnet
QRIO	-	Quest of cuRIOsity, Sony
R	-	Red component in RGB color space
R.U.R.	-	Rossum's Universal Robots
R'	-	Red component in normalized RGB color space
RGB	-	Red, Green, Blue space color
RIA	-	Robotics Industries Association
ROBIN	-	Robotics and Intelligent Systems
ROM	-	Read Only Memory
SDK	-	Software Development Kit
SDP	-	Service Discovery Protocol
SE	-	Sonny Ericsson

SIG	-	Special Interest Group
SIGCHI	-	Special Interest Group on Computer-Human Interaction
SMS	-	Short Message Service
TCS	-	Telephony Control System
TFT	-	Thin-Film Transistor
UI	-	User Interface
UML	-	Unified Modeling Language
VGA	-	Video Graphics Array
VM	-	Virtual Machine
VR	-	Variable reluctance
WAP	-	Wireless Application Protocol
WISM	-	Wireless LAN modules
WPAN	-	Wireless Personal Area Network
WTK	-	Wireless Toolkit
Y	-	Luminance (or brightness) component in Y′CbCr color space
Y′CbCr	-	One of two primary color spaces used to represent digital component video (the other is RGB)
YUV	-	Color space predecessor of Y′CbCr

Chapter 1

Introduction

Studies in the area of robotics have been growing swiftly for the past decade and the development in the field is even faster after the turn of this millennium. Previously, researches within the robotics community have been emphasizing more on technical challenges of achieving mobility, control and intelligence rather than social issues.

Lately, the development of robotics has motivated the feasibility of integrating robots into human daily lives. Therefore, study on human-robot interaction (HRI) has been taken to more serious consideration in research within the robotics community. As robots have to interact with humans, it is important to detect humans in the first place. The research presented in this thesis focuses on developing the basis for HRI system using face detection for mimic a natural communication between humans and robots.

Section 1.1 and 1.2 presents an introduction to Face Detection algorithms and robotics. The section 1.3 describes the main motivation of the thesis. Next, on section 1.4 the research background is presented. The approaches of this thesis are presented in section 1.5. The last section describes the outline of this thesis.

1.1 Introduction to Face Detection

Face detection is a process that determines whether there are any faces in an image. Face detection is not an easy process, as there are external and internal factors that affect the detection. A minor change in appearance, like wearing sunglasses or growing a mustache can make the task of face detection even more challenging. Also different illumination changes the color of faces significantly. There are several algorithms available in the literature that can help us to identify whether it is a face in an image. In a survey for Image Understanding, the Face Detection techniques are organized in two main categories (1; 2).

- Feature-based approach (3; 4)
- Image-based approach (5)

Feature-based approach requires prior information of the face. It makes an explicit use of facial features, which includes use edge information, skin color, motion and symmetry measures, feature analysis, deformable templates and point distribution.

Image-based approach does direct classification without any face knowledge derivation and analysis. It incorporates facial features implicitly into the system through training. Image based techniques include neural networks, linear subspace method like Eigenfaces (6), fisherfaces (7) etc. In Chapter 2 details of the classification of image-based approach and describe the methods that can be suitable to the project is given.

1.2 Introduction to Robotics

Robots are moving machines created by humans. The term “robot” originates from a play staged in London, called “Rossum’s Universal Robots”

(R.U.R.) written by the Czech writer, Karel Çapek, in 1921 (8). The word "Robot" is derived from the Slav *robota*, which means executive labor (8; 9).

In general, a robot can be defined as (10) "an automatic device that performs functions normally ascribed to humans or a machine in the form of a human". Robotics Industries Association (RIA)'s (8) defined a robot as a reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or special devices through variable programmed motions for the performance of a variety of tasks. Arkim (11) provided another working definition of robots, which is inclusive of mobile robots. According to him, an intelligent robot is "a machine, which is able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner".

Human and robots have strong correlation ever since the emergence of the very first idea of robotics. All robots fulfill one purpose, which is to serve humankind. Until today, humans have not observed any robot that causes intentional destruction to humankind. In fact, robots have aided human in many ways. Robots provide support, companionship, entertainment, etc. Eventually, study in robotics stirs towards HRI, as more robots assimilate into human daily life. The levels of HRI are described in section 2.3 on Chapter 2.

1.3 Main motivation of the Thesis

The main motivation of this thesis is the development of a face detection system on a mobile phone for achieving the appealing, yet often controversial contact human-computer. Along the years we keep finding new challenges and far more contact with new technology. The development of mobile phone technologies during the last decade has made possible using these portable computers for other tasks more than just calling. In addition the accessibility and price of the mobile phone creates a new area of development. Programming

intelligent systems on a mobile phone is still a young field. A human computer interaction (HCI) is probably the future we are about to approach, and using the technology that surround us will be the next step. To achieve the communication between humans and robots, I will implement face detection algorithm on a mobile robot.

Face detection plays an important role in today's world. However, research in this field is still young. Developing an algorithm for face detection in a small device such as a mobile phone can be useful in other areas for real world application, like human-computer interface, security, authentication for security of sensitive information, etc.

1.4 Research Background

A robot needs to "see" a human in the robotic environment. Thought machine vision, knowledge of the environment is acquired and processed by a robot and thus, providing a certain level of intelligence to the robot for the HRI (12). Over the past few years, face processing has also emerged as an important approach for HRI. Face processing methods includes face detection, face localization, face tracking, face recognition and facial expression recognition.

In order to proceed to the mentioned face processing, it is crucial to have good face detection ahead of it. Human face detection is taken as a natural representation and visualization of a human existence in the robot's environment. According to Marsic (13), the face to be "seen" by the computer may be complex but it is desirable to establish a natural *face-to-face* communication between human and robots. Through face detection, a human does not have to speak or manipulate the robot physically in order for robot to detect human presence. Therefore, face detection is an essential ability to have in HRI robots.

1.5 Approach

The study starts by sketching out the basic ideas underlying the communication *face to face* between humans and robots. I create a system for face detection on a mobile robot, which represents "the eyes" of a robot on HRI.

For the creation of this system I need a programmable portable camera, which can analyze and communicate information to a robot. A mobile phone camera was used for this project due to the accessible price and convenience of the technologies integrated on the device such as Bluetooth and Java.

In the research group of Robotics and Intelligent Systems (ROBIN) at the University of Oslo, there are different projects that use mobile phone cameras to look around in their environment. One example is the mBot shown in Figure 1-1. This robot will send via 3G or Bluetooth the image it sees to a server. To continue within this field of research I am implementing face detection to the image received by the mobile phone.

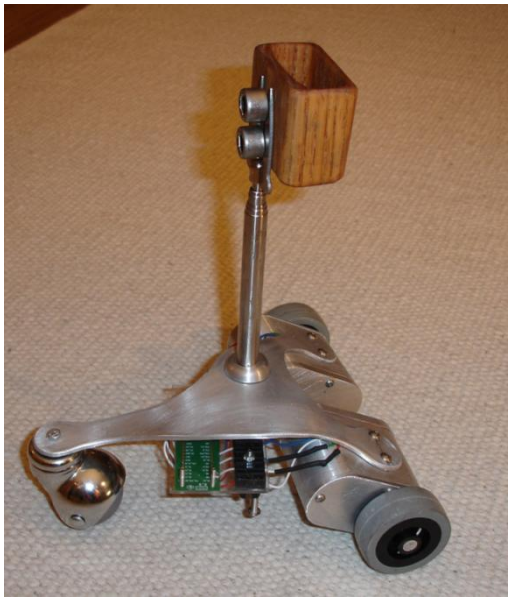

Figure 1-1 MES mBot robot using mobile phone camera

The algorithm starts with an image-processing step, consisting of identifying an area of interest that fulfills the defined criteria. The next step is

finding the (x, y) coordinate of the center of the area, which will be sent to the robot controller implementation. Finally, the robot controller generates the commands for motion which are sent via Bluetooth to the EZURiO developer kit (Blu2) antenna. The goal is to move the robot until this (x, y) equals the center of the picture taken with the mobile phone camera.

There are several algorithms available in the literature that can solve the problem of detecting faces. Face segmentation makes use of facial features in order to identify the face (5). Some algorithms for tracking face contours are known to be effective, but using a skin segmentation to reduce search space can be considered a reliable implementation for face detection (14).

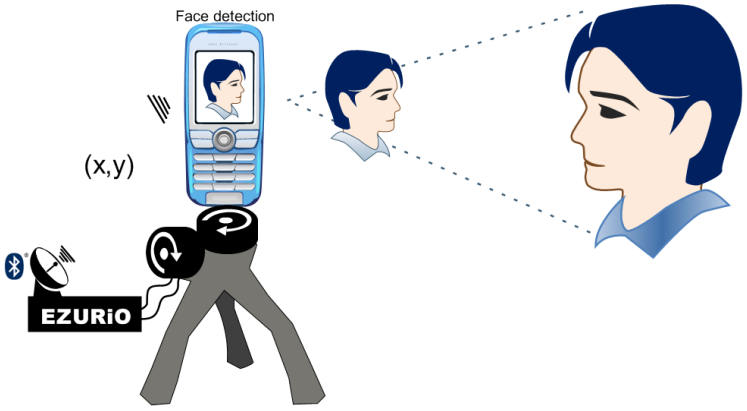

Figure 1-2 System for face detection on a motion robot

Human skin is relatively easy to detect in controlled environments, but detection in uncontrolled settings is still an open problem (4; 15). Many approaches to face detection are only applicable to static images assumed to contain a single face in a particular part of the image. Additional assumptions are placed on pose, lighting, and facial expression. When confronted with a scene containing an unknown number of faces, at unknown locations, they are prone to high false detection rates and computational inefficiency. Real-world images have many sources of corruption (noise, background activity, and lighting variation) where objects of interest, such as people, may only appear at low resolution. The problem of reliably and efficiently detecting human faces is

attracting considerable interest. An earlier generation of such a system has already been used for the purpose of flower identification by (16; 17).

The system proposed analyzes and implements different approaches for defining the area of interest. The first approach is finding a dark area on a white background (section 5.3.2), the second approach considers motion as a factor for finding a human (section 5.3.3) and finally the skin detection approach, which will find skin colored areas, identify the biggest of those areas and in that way identifying a face in an image (section 5.3.4, Figure 1-2).

Face detection in Java 2 Micro Edition (J2ME) is a challenge to develop, due to limited computing power and resources that a mobile phone can provide. In Chapter 2 I present an investigation of a variety of existing face detection approaches, to identify what is most suitable for the sort of images/resolution/processing available on the phone; then I will implement and evaluate that approach.

1.6 Outline

Chapter 2 contains the research work done for the face detection, a discussion of the different approaches is presented and human-computer interaction is described by their level of interaction.

Chapter 3 presents a brief introduction to the J2ME technology. The phone used to test the application is introduced and the development tools I used are described. Bluetooth architecture is introduced. EZURiO antenna and Step Motor functionality are described.

Chapter 4 provides a definition of color spaces as well as their use. By implementing my own test of skin detection in Matlab, it is shown how to

identify skin colored areas with limits of chrominance. Finally, I present the programming structure for improving efficiency and flexibility.

Chapter 5 contains code samples and explanations on how the application is developed. The different approaches are presented in this chapter and results of their performance are compared.

Chapter 6 presents a conclusion, a brief overview of the tested approaches and proposal for further work.

Chapter 2

Literature Survey

This chapter presents a literature survey of face detection. Different methods for detecting faces in images are presented. A discussion of implementation is done and the suggestion of a suitable approach is presented. An overview of human robot interaction is mentioned with their levels of interaction.

2.1 Face Detection Algorithms

Development of face detection on a mobile robot for human-robot interaction involves multidisciplinary studies because the project requires knowledge in image processing and robotics. In Chapter 1, it was specified that the goal of the study is to develop a face detection system on a mobile phone for potential HRI application. In order to achieve this goal, background studies and related works on face detection and HRI were done. As we can see on Figure 2-1, these areas of study are related to each other.

The core study of this work is face detection. The face detection is a subset of HRI. The mobile robot's face detection capability and its reaction during HRI are viewed as intelligent behaviors. Therefore, HRI is the subset of the intelligent robot family.

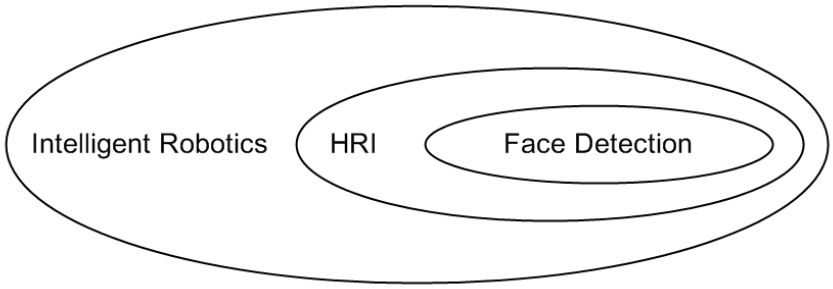

Figure 2-1 General overview of studies for face detection based HRI

Related work on face detection is discussed in section 2.2. The challenges, techniques, comparisons and issues of face detection will be given in this section. Majority of the study is based on image processing research. Later, different levels of robotics are introduced to provide an overview of the current work in that area.

2.2 Face Detection, Literature Survey

Human face detected in an image represents the presence of a human in a place. Evidently, face detection is the first step towards creating an automated system, which may involve other face processing. A difference between face detection and other face processing has been described by different authors (2; 18; 19), given by the following:

a. Face detection: To determine if there is any face in an image.
b. Face localization: To locate position of a face in image.
c. Face tracking: To continuously detect location of a face in image sequence in real-time.
d. Face recognition: To compare an input image against the database and report a match if similar.
e. Face authentication: To verify the claim of the identified individual in a given input image.
f. Facial expression recognition: To identify the state/ emotion of a human based on face evaluation.

g. Facial feature detection: To detect presence and location of face features.

Face detection remains an open problem. As introduced in Chapter 1 section 1.1, face detection is a process that determines whether there are any faces in an image. Many researchers have proposed different methods addressing the problem of face detection. As introduced the techniques are classified in to feature based and image based. In the following section we give a description of each method.

2.2.1 Feature Based Algorithms

In feature based analysis, visual features are organized into a more global concept of face and facial features. According to Hjelmås (1), the feature-based algorithms are divided in *low-level*, *feature* and *active shape analysis*. The methods researched for this thesis are low-level analysis.

2.2.1.1 Color Segmentation Algorithms

There are several color-segmentation algorithms available, which are effective for face detection. Some of them are presented below.

Face detection algorithms based on "Skin Color" has been an area of research during the last years (4; 20; 21). Three color spaces, RGB, Y'CbCr and HSI are of main concern for fulfilling this task. The comparison of the algorithms based on these color spaces is presented in (21) and have combined these to get a new skin-color based face-detection algorithm that improves accuracy. Experimental results (21) show that the proposed algorithm is good to localize a human skin color in an image with an accuracy shown on Table 2-1.

Table 2-1 Accuracy of the Skin and Non Skin patterns on various color transformation (21)

Color Space	HSV	CrCb	Y'CbCr	Nonlinear combination of Cr, Cb	Norm RGB
Skin	81.18	97.40	93.18	97.86	96.86
Non Skin	82.62	72.00	81.64	74.55	83.40

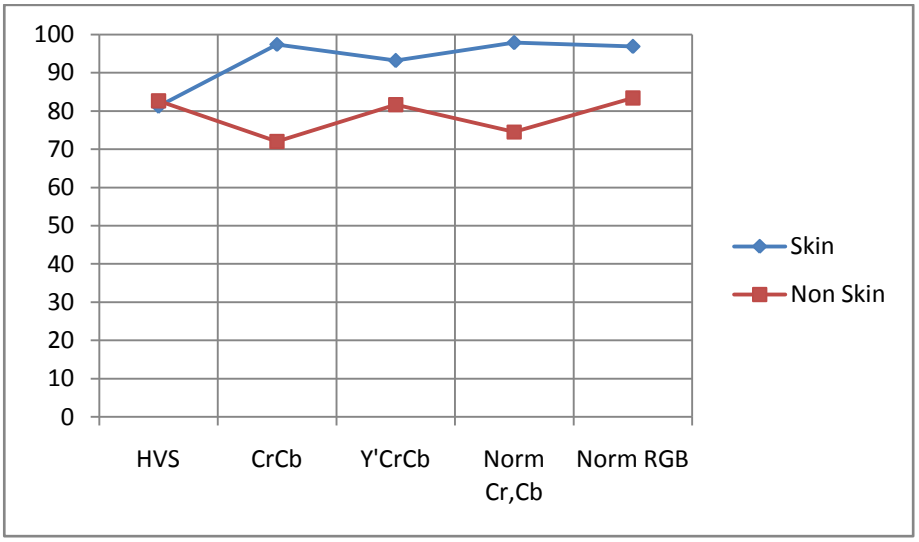

Figure 2-2 Accuracy in classification of skin and non skin

Another face detection algorithm uses color images in the presence of varying lighting conditions and complex backgrounds (3). The method detects skin regions over the entire image, and then generates face candidates based on the spatial arrangement of these skin patches. The algorithm constructs eye, mouth, and boundary by using a transfer of color space from RGB to Y'CbCr maps for verifying each face candidate.

2.2.1.2 Edge Detection Algorithms

Edge detection identifies outlines of an object and boundaries between objects and the background in the image. The goal is to mark the points at which the luminous intensity changes sharply. These sharp changes in images usually reflect important events.

The Roberts' Cross algorithm (22) performs is an edge detection algorithm that performs a two dimensional spatial gradient measurement on the image. The idea is to bring out the horizontal and vertical edges individually of the image and then to put them together for the resulting edge detection. This method is highly susceptible to noise.

2.2.1.3 Gray Scale Algorithms

This gray-scale algorithm was suggested by Yang and Huang (19), who observed that when the resolution of a face image is reduced gradually either by sub sampling or averaging, macroscopic features of the face will disappear and that at low resolution, face region will become uniform. This method consists of three levels, where the two highest levels are based on mosaic images at different resolutions and the lower level is improved edge detection.

This method is efficient for finding faces in complex backgrounds, when the face size and number of faces is unknown. It works with black and white images and is a flexible method for face recognition.

2.2.1.4 Motion Approach

The problem of face detection in still images is more challenging and difficult when compared to the problem of face detection in image sequence, since motion information can lead to probable regions where a person could be located. On the other hand, the results can be mistaken with other regions in the

image that move. For example in the case when the camera is mobile, we can encounter moving background.

The method of finding image is a feature-base approach. This finds features such as image edges, corners and other structures well localized in two dimensions. Firstly, the features are found in two or more consecutive images and after these features are matched between the frames. The algorithm eliminates the unimportant parts and creates an area of interest on the motion vectors. Alternatively, the features in one frame can be used as seed points at which to use other methods (for example, skin color detection on motion parts).

Motion is easy to implement. But alone can be hard to identify where the face of the human is.

2.2.2 Image Based Algorithms

The image base algorithms are known for their robustness for processing face detection (1). An example of this is the method proposed by Rowley (23), which uses neural networks. However, the limited computing capacity we are working with for this project, require that we narrow the research to methods that are able to run on the target platform. Below we present the methods we consider suitable for implementation on this project.

2.2.2.1 Template Matching Algorithms

Template matching is a technique for finding small parts of an image which match a template. For face detection it can be used an oval template for matching the shape of the head; and eliminate possible candidates that fit other criteria. For example, if it is first found all skin colored areas, and then match them to the template; it could be possible to find a face. In that case arms, hands, legs or other skin colored areas will be rejected as not-face area.

Cross correlation is a template matching algorithm that estimates the correlation between two shapes that have a similar orientation and scale (24). It is quite robust to noise, and can be normalized to allow pattern matching independently of brightness and offset in the images.

We find the cross-correlation algorithm to be of limited utility due to its assumption on geometric scale and orientation of the templates. Particularly this method for face detection is resource consuming. For the concern of this thesis we will rather use a method that increases the performance of the analysis.

2.2.2.2 Holistic Approach

In holistic methods, the face is taken as input data. One of the main algorithms that fall under this category is the eigenface method.

Eigenface method (6) is based on the implementation of Principal Component Analysis (PCA) over images. To generate a set of eigenfaces, a large set of digitalized images of humans are taken under the same lighting conditions. The eyes and mouths are lined up and then the mean value of each pixel is extracted with use of the mathematical tool PCA. A matrix of covariance with the mean image is created. Finally, a calculation of eigenvectors and eigenvalues are done to the matrix to find the principal components of the image. This technique is considered the first facial recognition technology. It has also been used for handwriting analysis, lip reading, voice recognition and medical imaging. Other examples of holistic methods are fisher faces and support vector machines (6; 7).

2.2.2.3 Hybrid Approach

The idea of this method comes from how human vision system sees both face and local features. The method proposed by Wang (15), combines both sound and picture analysis. The person initially talks and is estimated the

positions by using a microphone. After, the motion analysis estimates where the face can be. Some other examples of the hybrid approach are modular eigenfaces (25) and component-based methods (26).

2.2.3 Discussion for Face Detection

The main advantage of the feature-based methods (section 2.2.1), is the simplicity and intuitiveness of the classification rules. These methods are applicable for systems where color or motion is available.

Image-based approaches (section 2.2.2) are known for their high performance. Still, these methods are complex and require of training processes for improving results.

To increase the performance of the search for faces on an image, the best is to combine more than one method. A recent paper proposes machine learning algorithms to find suitable color space and simple decision rules (27), the method shows a way to overcome these difficulties. However, the limited resources we have on the mobile phone (section 3.2.5) limit our possibilities.

For this thesis we propose to use feature-based techniques. The image base algorithms require bigger storage space and a representative training dataset. Nevertheless, even though there is wide range of algorithms available for face detection. Tuning these algorithms on to our J2ME system will be a challenge.

2.3 Human-Robot Interaction (HRI)

Today, HRI has received much attention in robotics. According to Scholtz (28; 29), HRI is a cross disciplinary area, that involves psychology, sociology, cognitive science, communication and robotics. HRI refers to how

human and robot interact in a given situation. HRI has been a fraction of robotics research but it is still in its infancy as researches are more focused on achieving better control and mobility in robots (30) to create an interaction environment. If a robot is in direct contact with a human, it must have some natural means of communication (31). Therefore, speech and vision-based HRI are more widely used for natural HRI development as no equipments or tools are needed in between human and robot during the interaction process.

Human-Robot interaction currently takes many forms. Dangerous task such as urban search and rescue (32; 33) and hazardous material clean up (34) require a human operator to be removed from the physical location of the robot. The robots that assist the elderly and handicapped share the same physical space with their users, often transporting them through the world. Others, such as Sony's AIBO[1] and QIRO[2] (Figure 2-3) provide entertainment and companionship for people.

The HRI is a subset of the field of human-computer interaction (HCI). HCI has been defined in many ways. One example is the definition used by the Curriculum Development Group of the Association for Computing Machinery (ACM) Special Interest Group on Computer-Human Interaction (SIGCHI) (35): "Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them". Since robots are computing intensive systems designed to benefit humans, we feel that HRI can be informed by the research in HCI.

[1] AIBO "Artificial Intelligence roBOt", homonymous with "companion" in Japanese
[2] QRIO "Quest for cuRIOsity", originally named Sony Dream Robot or SDR

Figure 2-3 Sony AIBO and QRIO

2.3.1 Levels of Human-Robot Interaction

Scholtz (36) proposes three levels of human interactions that are possible. These levels are schematized in Figure 2-4. In the first, supervisory interactions take place between a human and a robot in a remote location. The supervisor needs to know the mission, an overview of the situation, the capabilities of the robot and any problems of how the robot interacts with other robots (if there are any). Scholtz points out that this interaction level is similar to the HCI domain of complex monitoring devices.

The second level of interaction is *peer to peer*, where each human and robot contributes to a team according to his/its capabilities. In this situation, the human user will need to know the status of the robot, the robot's world model, other interactions that are occurring, and the robot's action capabilities.

The final level of interaction is mechanic, where a user is teleoperating a robot, requiring the user to be a skilled user of the robot. In this level, the user needs to know similar things to the peer-to-peer level, but must also have information about the robot's sensors, other jobs that need attention, effects of adjustments on plans and other interactions, and mission overview and timing constraints. This interaction level has several drawbacks, including the need for

high bandwidth communication, cognitive fatigue from repetitive task and information overload (37).

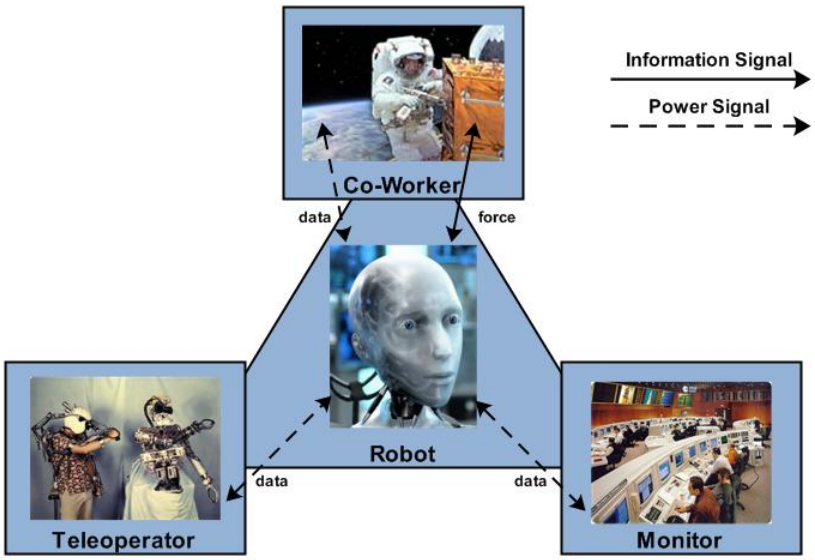

Figure 2-4 Levels of Human Interaction by Scholtz

2.4 Discussion for HRI

The HRI (section 2.3), presented the different levels of interaction that a human and a robot can have. This thesis proposes an implementation of the second level of HRI proposed by Scholtz. The algorithm will let the robot do its task, after will check the results and compare them to what we can actually see.

Intelligent robotics is the outer core of the Figure 2-1. This is just a representation to what we could have been done if the resources and time were on our favor. Nevertheless, the developing of new technologies and the new interests on research on robotics will let the future researcher create Artificial Intelligence systems on portable mobile phones.

Chapter 3

System Development Technology

3.1 Technology to Use

This chapter gives and insight to the technologies used as J2ME, Bluetooth, Wireless communication and controller of a stepper motor robot. This chapter can be used as reference for future readers. The section 3.2 is cited from the documentation and website of Sun Microsystems (38). This citation was needed for the importance of the background. This chapter can be used as reference for further work.

3.2 Java™ 2 Micro Edition (J2ME)

J2ME enables Java applications to run on devices with limited resources such as Personal Digital Assistants (PDA), mobile phone, interactive pager, etc. Being very much different from Java 2 Enterprise Edition (J2EE)/Java 2 Standard Edition (J2SE), the J2ME architecture is designed to be flexible,

modular and scalable in order to meet the market demand. This modularity and scalability are defined by J2ME technology in a model with three layers (38) of software built upon *Host Operating System* of the devices. These layers are illustrated in Figure 3-1 and explained bellow the image.

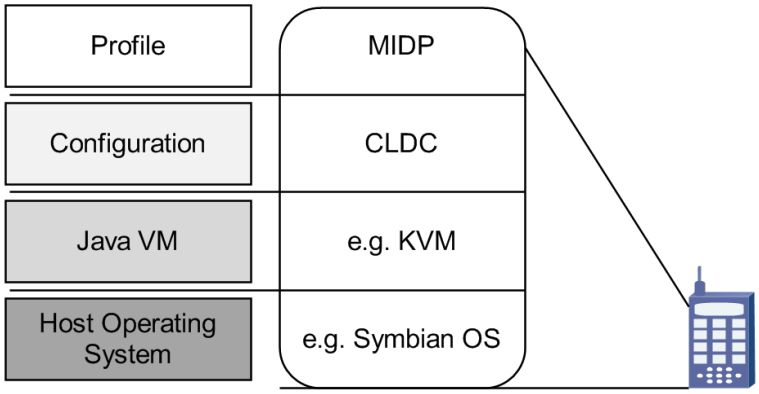

Figure 3-1 High level view of J2ME

- *Java Virtual Machine Layer.* This layer is implementation of a Java virtual machine (JVM) that is customized for a particular device's host operating system and supports a particular J2ME configuration
- *Configuration Layer.* The configuration layer is less visible to users, but is very important for profile implements. It defines the minimum set of Java virtual machine features and Java class libraries available on a particular category of devices representing a particular market segment. In a way, a configuration defines the "lowest common denominator" of the Java platform features and libraries that the developers can assume to be available on all devices of the same category.
- *Profile Layer.* A profile is layered on top of the configuration. This is the most visible layer to users and application providers. It defines the minimum set of Application Programming Interfaces (API) available on a particular "family" of devices representing a particular market segment, for instance washing machine thus portable to any device that supports that profile. Applications are written for a particular profile and are thus portable to any device that supports that profile. A device

can support multiple profiles. The mobile phone specific profile is defined under the name of Mobile Information Device Profile (MIDP).

Today's mobile phones use Mobile Information Device Profile (MIDP) combined with Connected Limited Device Configuration (CLDC) as the J2ME run-time environment.

3.2.1 Configurations

A Configuration defines the basic J2ME runtime environment. This environment includes de K Virtual Machine (KVM) or other conformable VM which is more limited than the VM used in the standard edition of Java. Currently, two configurations are defined in J2ME (38).

- **Connected, Limited Device Configuration (CLDC).** CLDC is aimed at the low end of the consumer electronics range. The CLDC defines targeted Java platforms, which are small resource-constrained devices, each with a memory budget in the range of 160kB to 512kB (39). This configuration includes some new classes (shown in Table 3-1) designed specially to fit the needs of small-footprint devices.
- **Connected Device Configuration (CDC).** CDC addresses the needs of shared, fixed, connected information devices such as TV set-top boxes and web-screen phones that lie between those addressed by the CLDC and the full desktop system running J2SE. These devices have more memory (typically more than 2 MB) and processors that are more capable, hence they can support much more complete Java software environment. To ensure upward compatibility between configurations, the CDC shall be a superset of the CLDC.

The target device of this thesis is a mobile phone, which will be described on the section 3.2.5 is using CLDC packages. We developed a client application on the top of the MIDP with extends the CLDC. Although the CLDC

layer is not as visible as the MIDP layer to the application programmers, it is essential to understand the CLDC first because it defines the significant differences between J2ME and the well-accustomed J2SE environment.

Table 3-1 CLDC packages

Package	Provides
`java.io`	Provides classes for input and output through data streams
`java.lang`	Provides classes that are fundamental to the Java programming language
`java.lang.ref`	Provides support for weak references
`java.util`	Contains the collection classes, and the date and time facilities
`javax.microedition.io`	Classes for the GCF

3.2.2 CLDC Requirements and Specification

The minimum total memory budget required by a KVM implementation is about 160 kB, including the virtual machine, the minimum Java class libraries specified by the configuration, and some heap space for running Java applications. A more typical implementation requires a total memory budget of 256 kB, of which half is used as heap space for applications, 40 to 80 kB is needed for the virtual machine itself, and the rest is reserved for configuration and profile class libraries. The ratio between volatile memory (e.g., DRAM) and non-volatile memory (e.g., ROM or Flash) in the total memory budget varies considerably depending on the implementation, the device, the configuration and the profile. A simple KVM implementation without system class pre-

linking support needs more volatile memory than a KVM implementation with system classes pre-located into the device (39).

In order to support a Java runtime environment with such limited resources, the CLDC defines reduced requirements for the virtual machine and the Java language specification. Compared to J2SE, the differences are as follows:

- **No object finalization and no weak references.** `Object. Finalize ()` does not exist.
- **Limitations on error handling.** Most subclasses of `java.lang.Error` are not supported. Runtime errors are handled in an implementation-dependent fashion.
- **No support for the Java Native Interface (JNI) or reflection features.** In particular, there is no support for object serialization.
- **No user-defined class loaders.** An application cannot influence how classes are loaded. Only the runtime system can define and provide class loaders.
- **Class file verification is done differently.** The standard class verification process is too memory-consuming for small devices, so an alternate process was defined. The most of the verification work is separated into a *pre-verification* step that is typically performed on a server or desktop system before the class file is downloaded to the device. The pre-verified class flies are then processed on the device using a much simpler kind of verification that merely validates the result of the pre-verification step.

The CLDC includes pretty much limited number of classes and interfaces from java.lang, java.io and java.util. Besides, the CLDC defines the classes that make up the Generic Connection Framework or GCF for short. With the GCF, all communication is abstracted through a set of well-defined

interfaces. For example, to open a Web connection, a developer might simply write:

```
Connector.open ("http://www.example.org");
```

The GCF is leveraged and extended by the MIDP to allow the creation of network-aware applications. Now that a basic understanding of J2ME is attained, we step forward to the MIDP and MIDlets technology.

3.2.3 MIDP – Java on Mobile Phone

The MIDP specification was defined through the Java Community Process (JCP) by an expert group of more than 50 companies, including leading device manufactures, wireless carriers, and vendors of mobile software. It defines a platform for dynamically and securely deploying optimized, graphical, networked applications. The mobile information device profile is shown in Figure 3-2.

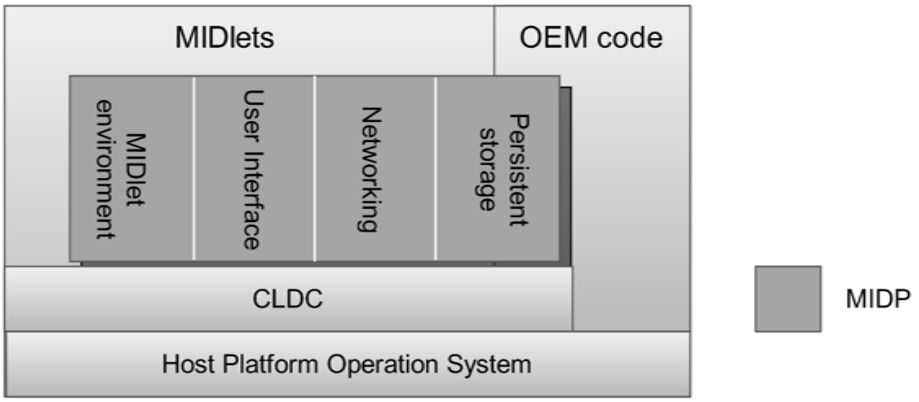

Figure 3-2 the Mobile Information Device Profile

Developers using the MIDP can write applications and deploy them quickly to a wide variety of mobile information devices. The MIDP has been widely adopted as the platform of choice for mobile applications. It is deployed globally on millions of phones and PDAs, and is supported by leading integrated development environments (IDE).

Companies around the world have already taken advantage of the MIDP to write a broad range of consumer and enterprise mobile applications. Today the MIDP 2.0 is implemented on many devices. It is a revised version of the older MIDO 1.0 specification, and includes new features such as an enhanced user interface, multimedia and game functionality, grater connectivity, over-the-air (OTA) provisioning, and end-to-end security. The MIDP 2.0 is backward compatible with the MIDP 1.0, and continues to target Mobile Information Devices (MIDs) such as mobile phones and PDAs.

3.2.4 MIDlets and MIDlets Suite

Java applications that run on the MIDP are known as MIDlets, and in some ways resemble J2SE concept of Applets. Like Applets, MIDlets must extend the MIDP-defined abstract class javax.microedition.MIDlet and provide a public default constructor, which enables the system software to create an instance of MIDlet. MIDlets run in an execution environment within the Java VM that provides a well-defined lifecycle controlled via MIDlet class methods, which each MIDlet must implement. This MIDlet lifecycle is examined in the following section.

A collection of one or more MIDlets is packaged together into one JAR (Java archive) file to form a MIDlet suite. All the MIDlets in a suite share both static and runtime resources of their host environment, e.g. classes loaded into their mutual Java VM and persistent storage; in the MIDP 2.0 though, inter-suite access is allowed as for the persistent storage, if explicit permission is given.

3.2.4.1 MIDlets Life cycle

MIDlet has one more analogy to Applet: like the Applet's start, stop and destroy methods, the MIDlet class defines three abstract methods that the system software calls to start, pause and destroy an application.

On a mobile information device (MID), an Application Management Software (AMS) controls the activation and deactivation of MIDlets. The AMS also maintains state information about each MIDlet. As Figure 3-3 shows, there are only three states possible:

- *Active* – the application is running.
- *Paused* – the application has yet to run or is in an idle state.
- *Destroyed* – the application is terminated

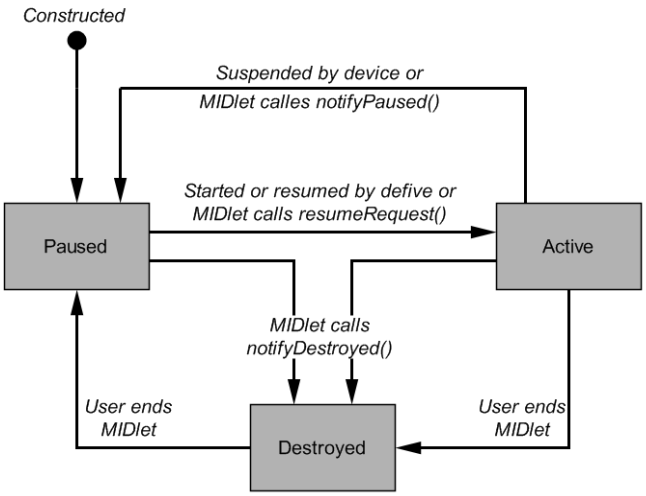

Figure 3-3 MIDlet life cycle

The destroyed state indicates that a MIDlet has terminated and resource associated with it can be freed by the system. Once in the destroyed state, the MIDlet cannot transition back to the other state. Again, developers are to implement the destroyApp method in a proper way so that the important data are saved, all the allocated resources are released, any background threads are terminated and any active timers are stopped as well as all the connections are closed. Remember, finalizes do not exist in the CLDC-based profiles, so the only way to free the resource used by an object is to do it explicitly.

3.2.4.2 MIDlet Packing

MIDlets need to be properly packaged before they can be delivered to a device for installation. A single JAR file must contain *all* the required class files and any images (e.g. icons), or other files to which the MIDlet needs access at runtime. Apart from CLDC and MIDP classes and any vendor-specific classes, the JAR file must be complete in itself. The reason for that is the prohibition imposed by the MIDP on the class file sharing between suites or dynamic downloading and installation of new classes as an application runs. Packing information that tells the device what is in the JAR must be supplied in the JAR's manifest file. The manifest file defines important information about MIDlet, such as the name, main class, and icon as well as information about vendor and required profile and configuration versions.

The JAR file is accompanied by an external file called the *Java Application Descriptor* (or JAD file). It is similar to the manifest; in fact, the two files share some data in common. But more importantly, the JAD file gives the information that helps the decision whether to download the complete JAR file and install or not. This information includes the size of the JAR file, the minimum amount of persistent memory required by the application, the version number, and so on. Placing this information in a separate file instead of in the JAR enables the device to quickly download the JAD file for analysis by the MIDP installation software.

3.2.5 Target Device SE Z520i

To accomplish this thesis we are using a Sony Ericsson Z520. The main technical specification of this device is summarized in Table 3-2 Sony Ericsson Z520 technical specifications (40). The SE Z520 is a quad-band, clamshell phone with both internal and external screens.

Worth to mention is that there is no limit set to the maximum JAR size. Moreover, the Bluetooth power class 2, that is using maximum 4dBm radio link, which operates in the globally available 2.4 GHz radio frequency band, ensures fast and secure communication up to a range of 10 meters. This is essential for the communication between camera and robot.

Table 3-2 Sony Ericsson Z520 technical specifications

Vendor	Sony Ericsson
Operating System	Sony-Ericsson OS
Developer Platform	MIDP/2.0, JTWI/1.0
Configuration	CLDC/1.1

Java Technology	• JSR 139 CLDC 1.1
	• JSR 118 MIDP 2.0
	• JSR 120 Wireless Msg API
	• JSR 75 File and PIM API
	• Java™ API for Bluetooth (JSR 82)
	• The following functions for the JSR 135 Mobile Media API:
	  o Audio playback
	  o Video playback
	  o Camera snapshot

Memory	Up to 16 MB (depending on software configuration/file content)
Max JAR Size	Memory allocated dynamically
Network Data Support	GPRS
Band Functionality	GSM (Global System for Mobile Communications)
Bluetooth	Bluetooth Specification 2.0
Screen Display	• Type: Full graphical • Resolution: 128 x 160 pixels

	• Technology: TFT • Colors displayed together: 65K (16 bit) • Backlight color: White
Picture sizes (resolution) VGA Camera	VGA (640 x 480 pixels) QVGA (320 x 240 pixels) QQVGA (160 x 120 pixels) Extended (1289 x 960 pixels) - TBD
Physical Measurements	• Dimensions: 93 x 46 x 24 mm • Weight: 96g
Keypad Description	Includes 4-way navigation with select key, dedicated internet key, +/- volume keys, side camera key, and two soft buttons
Browser	WAP 2.0, XHTML

Figure 3-4 Sony Ericsson Z520 device

3.2.6 Utility Tools

3.2.6.1 Wireless Toolkit

The J2ME Wireless Toolkit, or WTK for short, consists of sets of tools that provide application developers with the emulation environments, documentation and examples needed to develop application target at MIDP compliant mobile phones.

The Sun Java Wireless Toolkit 2.5.1 for CLDC is a state-of-the-art toolbox for developing wireless applications that are based on J2ME's Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). Once you have written your MIDlet code on a text editor or on an IDE, the WTK would be the first choice of tool to test the code on its emulator. Besides emulating, you can also build, package and sign the MIDlet suites. The lack of emulation on device made me look for other options.

Figure 3-5 Connection proxy SE SDK

For this thesis and since we are using a SE mobile phone it makes sense to use the Sony Ericsson SDK 2.2.4 for the Java ME Platform. In mainly works as the WTK of Sun but it has a bit more specific functions to the SE mobile phones. The major difference is in the SDK connection proxy (Figure 3-5) and phone side SDK agent which provide a more stable and efficient channel for interaction between the Java platform and SDK e.g. on-device debugging (VM) and device explorer. In essence, the SDK is a WTK 2.5.0 foundation, which is customized to better reflect the Sony Ericsson Java platform characteristics.

3.2.6.2 Eclipse with J2ME plug in

During the work with this thesis, some Java Integrated Development Environments (IDEs) have been explored. The Eclipse IDE is convincing due to its high usability. It is open-source and freely available. Using native GUI libraries, it gives the developer a much better look-and-feel than the other IDEs. Eclipse is a framework for Java development, meaning that its functionality can be extended by the use of plug-ins. One suitable plug-in was found for J2ME application development during the work with this thesis, the EclipseME plug-in (41). With the EclipseME plug-in installed, MIDlet suites can be created easily with the Eclipse IDE. It is also fully proved with the SE SDK and even though it has some small problems when running the emulator, for example problem to connect to device is not a big issue. The Eclipse SDK version 3.2.2 and EclipseME 1.6.8 plug in was used without major problems.

3.3 Bluetooth

Bluetooth is a standard for short range, low power, low cost wireless communication that uses radio technology. Ericsson Mobile Communications started developing the Bluetooth system in 1994, looking for a replacement to the cables connecting mobile phones and their accessories. Ericsson joined forces with Intel Corporation, International Business Machines Corporation (IBM), Nokia Corporation, and Toshiba Corporation to form the Bluetooth Special Interest Group (SIG) in early 1998. Over 2100 companies around the world already support the resulting Bluetooth specification, developed by Bluetooth SIG. Bluetooth technology. The Wireless Personal Area Network (WPAN) technology, based on the Bluetooth Specification, is now and IEEE standard under the denomination of 802.15 WPANs.

The Bluetooth system is named after a tenth-century Danish Viking king, Harald Blåtand, who united and controlled Norway and Denmark. The first Bluetooth devices hit the market around 1999.

The Bluetooth SIG is responsible for further developed of the Bluetooth standard. Sony Ericsson, Intel, IBM, Toshiba, Nokia, Microsoft, 3COM, and Motorola are some of the companies involved in the SIG. The composition of the Bluetooth SIG is one of the major strengths of the Bluetooth technology. The mixture of both noticeable software and hardware suppliers participating in the further development of the Bluetooth technology ensures that Bluetooth products are made available to end users. Microsoft supports Bluetooth in their Microsoft Windows Operating System (OS); hence, Bluetooth software is made available to the vast majority of the desktop software market. At the time of writing, Intel is including Bluetooth technology in several new main board chipsets, especially for laptop computers. Both Nokia and Sony Ericsson include Bluetooth technology in their latest mobile phone. This all adds up to a wide availability of the Bluetooth technology for end-users. Information of more commercial nature about the Bluetooth technology is available on the Bluetooth technology website.

This thesis project uses the Bluetooth Specification version 1.1, the Bluetooth version implemented in most mobile devices at the moment. However, the Bluetooth 1.2 specification is already completed and the Bluetooth 2.0 specification is in the works. At the time of writing, Enhanced Data Rate (EDR) Bluetooth has just been introduced by the Bluetooth SIG, raising the gross air data rate from 1 Mbps to 2 Mbps or 3 Mbps. Devices conforming to these new specifications will probably show up shortly after the completion of this Master thesis.

3.3.1 Bluetooth Architecture

The Bluetooth specification aims to allow Bluetooth devices from different manufacturers to work with one another, so it is not sufficient to specify just a radio system. Because of this, the Bluetooth specification does not only outline a radio system but a complete protocol stack to ensure that Bluetooth devices can discover each other, explore each other's services, and make use of these services.

The Bluetooth stack is made up of many layers, as shown in Table 3-3. The Host Controller Interface is usually the layer separating hardware from software and is implemented partially in software and hardware/firmware. The layers below are usually implemented in hardware and the layers above are usually implemented in software. Note that resource constrained devices such as Bluetooth headsets may be all functionally implemented in hardware/firmware.

Table 3-3 Description of Bluetooth protocol layers

Layer	Description
Applications	Bluetooth profiles guide developers on how applications should use the protocol stack
Telephony Control System (TCS)	Provides telephony services
Service Discovery Protocol (SDP)	Used for service discovery on remote Bluetooth devices
WAP and OBEX	Provide interfaces to higher layer parts of other communications protocols
RFCOMM	Provides an RS-232 like serial interface
Logical Link Control and Adaption Protocol (L2CAP)	Multiplexes data from higher layers and converts between different packet sizes
Host Controller Interface	Handles communication between the

	host and the Bluetooth module
Link manager Protocol	Controls and configures links to other devices
Baseband and Link Controller	Controls physical links, frequency hopping and assembling packets
Radio	Modules and demodulates data for transmission and reception on air

The interested readers will find further information about layers of the Bluetooth stack in the Bluetooth book by Bray and Sturman (42) and in the Bluetooth specification (43).

3.3.2 EZURiO Bluetooth Development Kit

The EZURiO Development Kit (44) is designed to support the development of hardware, applications and software for the EZURiO range of BISM II Bluetooth Intelligent Serial Modules and WISM Wireless LAN modules. The EZURiO development kit is available in five options to cover the widest range of wireless technologies, including hardware support for development of applications.

The development board allows the EZURiO Bluetooth or Wireless LAN Module to be connected to a PC or a mobile phone. This device will be the connection between the mobile phone and the robot.

Figure 3-6 EZURiO – Wireless Development Kit

3.3.2.1 EZURiO Terminal

The development board will connect to any Bluetooth module to the serial port of the other device, which can be also a virtual serial port. From the PC is possible to communicate with the module using the EZURiO Terminal application (Figure 3-7 EZURiO Terminal)

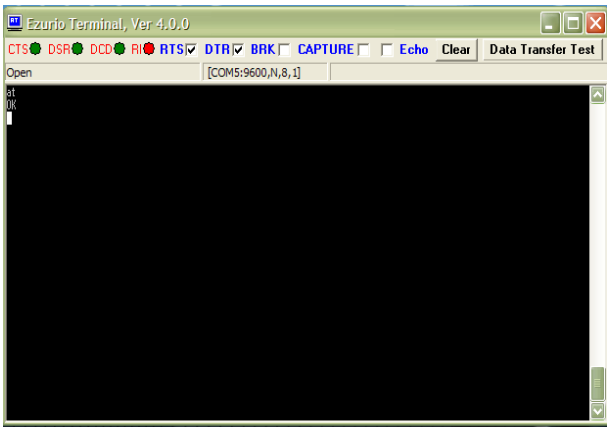

Figure 3-7 EZURiO Terminal

The lack of information around the use of this device caused some problems under developing. We will describe this problem on Chapter 5.

3.4 Stepper Motor

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the directions of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied.

A stepper motor consists of a set of coils that produce magnetic fields and interact with the fields of permanent magnets (PM), called the rotor. The different coils are switched on and off in a specific sequence to cause the motor shaft to turn through the desire angle. The motor can operate in either direction. When the coils of a stepper motor receive current, the rotor shaft turns to a certain position and then stays there unless or until different coils are energized. Unlike a conventional motor, the stepper motor resist external torque applied to the shaft once the shaft has come to rest with current applied.

A stepper motor's design is virtually identical to that of a low-speed synchronous AC motor. In that application, the motor is driven with two phases AC, one phase usually derived through a phase shifting capacitor. Another similar motor is the switched reluctance motor, which is a very large stepping motor with a reduced pole count, and generally closed-loop commutated.

3.4.1 Advantages

1. The rotation angle of the motor is proportional to the input pulse.
2. The motor has full torque at standstill (if the windings are energized)
3. Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next.
4. Excellent response to starting/ stopping/reversing.
5. Very reliable since there are no contact brushes in the motor. Therefore the life of the motor is simply dependant on the life of the bearing.
6. The motors response to digital input pulses provides open-loop control, making the motor simpler and less costly to control.
7. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.

8. A wide range of rotational speeds can be realized as the speed is proportional to the frequency of the input pulses.

3.4.2 Disadvantages

1. Resonances can occur if not properly controlled.
2. Not easy to operate at extremely high speeds.

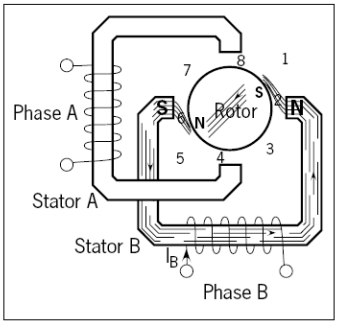

Figure 3-8 Magnetic flux through two-pole stepper motor

3.4.3 Phases, Poles and Stepping Angles

Usually stepper motors have two phases, but three- and five-phase motors also exist. A bipolar motor with two phases has one winding/phase and a unipolar motor has one winding, with a center tap per phase. Sometimes the unipolar stepper motor is referred to as a "four phase motor", even though it only has two phases.

Motors that have two separate windings per phase also exist—these can be driven in either bipolar or unipolar mode. A pole can be defined as one of the regions in a magnetized body where the magnetic flux density is concentrated. Both the rotor and the stator of a step motor have poles. In reality several more poles are added to both the rotor and stator structure in order to increase the number of steps per revolution of the motor, or in other words to provide a smaller basic (full step) stepping angle. The permanent magnet stepper motor

contains an equal number of rotor and stator pole pairs. Typically, the PM motor has 12 pole pairs. The stator has 12 pole pairs per phase. The hybrid type stepper motor has a rotor with teeth. The rotor is split into two parts, separated by a permanent magnet—making half of the teeth south poles and half north poles. The number of pole pairs is equal to the number of teeth on one of the rotor halves. The stator of a hybrid motor also has teeth to build up a higher number of equivalent poles (smaller pole pitch, number of equivalent poles = 360/teeth pitch) compared to the main poles, on which the winding coils are wound. It is the relationship between the number of rotor poles and the equivalent stator poles, and the number the number of phases that determines the full-step angle of a stepper motor. In Figure 3-8 we can see the magnetic flux path through a two-pole stepper motor, with a lag between the rotor and stator.

$$Step\ angle = \frac{360}{(N_{Pb} \times Ph)} = 360/N$$

$$N_{Pb} = Number\ of\ equivalent\ poles\ per\ phase = number\ of\ rotor\ pole$$

$$Ph = Number\ of\ Phases$$

$$N = Total\ number\ of\ poles\ for\ all\ phase\ together$$

3.5 ROBIN: Hardware & Software

The Robotics and Intelligent Systems Group (ROBIN) (45), and its robotic laboratory (46) consist of different types of robots. The robots are made by Mats Høvin. For this thesis it was designed a mobile phone holder with two stepper motors which gave the option of moving in a (x, y) axis.

The structure was designed with SolidWorks, a 3D mechanical CAD program that can print out in ABS-plastic with the ROBIN's 3D printer "Dimension SST768". The program used to control the printer is CatalystEX. The maximum structure size that this printer allows is 203x203x305 mm. The time used for producing is about 10 min/cm^2.

The stepper motor used for this robot is a bipolar stepper motor with a rotor made up of a permanent magnet from ELFA with number 42PM100S01. In Figure 3-9 we can see a picture of the first prototype of implementation. To connect this stepper motor we used a Dual Full-Bridge Motor Driver. For further details of the motor driver, Data Sheet can be consulted (47).

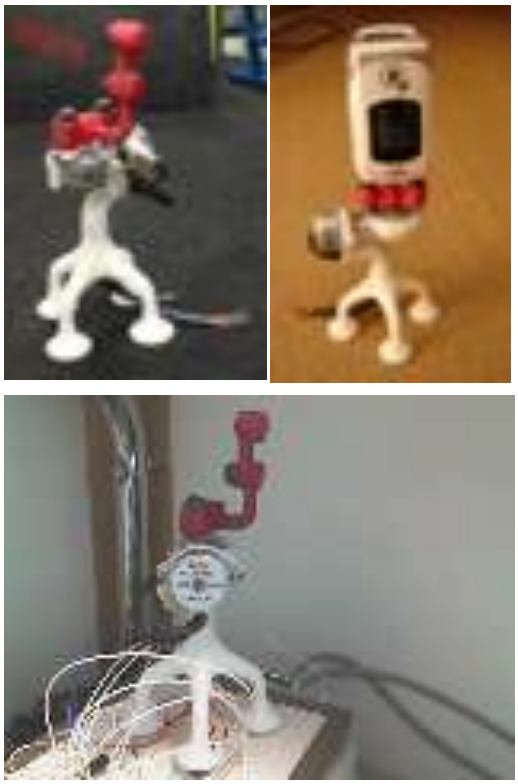

Figure 3-9 First prototype of implementation

The Motor Driver is connected to the output pins of the EZURiO antenna and the communication via Bluetooth is established with the phone. The connection protocol is explained in Chapter 5.

Chapter 4

Images and Features

This chapter gives an introduction to color spaces. Present the RGB and Y'CbCr color spaces and the conversion between them. My own implementation for skin detection using color spaces is presented in section 4.1.4. All calculations and data handling in this section were performed in Matlab. The section 4.2 will present the design architecture of the program for better efficiency and control of the developed program.

4.1 Color Space

A color space is a mathematical way of representing colors as data, typically as three or four values or color component (48; 49). Different color spaces have historically evolved for different applications. In each case, a color space was chosen for applicable reasons. A choice was made on a particular color space because the mathematical elements needed to process were simpler or faster. A certain choice was better because it required less storage and bandwidth on digital buses.

Whatever historical reasons caused color space choices in the past, the convergence of computers, the internet, and a wide variety of video devices, all using different color representations, is forcing the digital designer today to

convert between them. Converters are useful for a number of markets, including image processing and filtering. The converter's basic function is to convert form one color space to another. A long this section the color spaces RGB and Y'CbCr is described and the way they can convert to get a better result on skin color analysis. An implementation for skin color analysis is done in section 4.1.4.

4.1.1 RGB Color Space

RGB color space is a simple and robust color definition used in computer systems and the Internet to help ensure correct mapping of a color from one platform to another without significant loss of color information (49; 50; 9). RGB uses three numerical components to represent a color. This color space can be thought of as a three-dimensional coordinate system whose axes correspond to the three components, R or red, G or green, B or blue. RGB is the color space used by computer displays (Figure 4-1). RGB corresponds most closely to the behavior of the human eye (48).

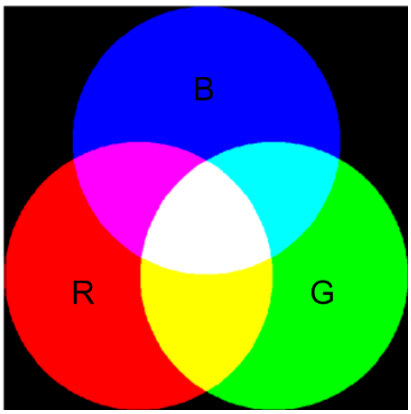

Figure 4-1 RGB color space

RGB is an additive color system. The three primary colors red, green and blue are added to form the desired color. Gamma-corrected values are noted R'G'B'. Each component has a range of 0 to 255 (for an 8-bit representation), with all three 0s producing black and all three 255s producing white. The Figure 4-2 shows the RGB color cube (on the left) with the eight corners.

Table 4-1 100% RGB Color Bars

	Normal Range	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
R	0 to 255	255	255	0	0	255	255	0	0
G	0 to 255	255	255	255	255	0	0	0	0
B	0 to 255	255	0	255	0	255	0	255	0

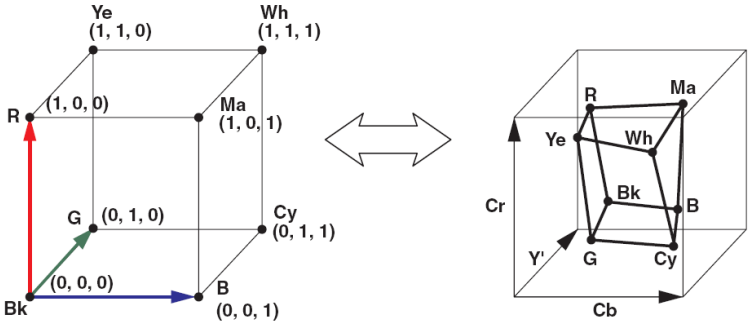

Figure 4-2 RGB and Y'CbCr color cubes (48)

4.1.2 Y'CbCr Color Space

Y'CbCr color space was developed as part of the Recommendation ITU-R BT.601 for worldwide digital component video standard and is used in television transmission. Y'CbCr is a version of the YUV color space where Y represents luminance (or brightness), U represents color, and V represents the saturation value. Here the RGB color space is separated into a luminance part (Y') and two chrominance parts (Cb and Cr).

The historical reason for this choice, over R'G'B' were to reduce storage and bandwidth. Since the eye is more sensitive to change in brightness than change in color, the reduction in bandwidth requirement seemed a valid trade for little or no visual difference.

To generate the same color in the RGB format, all three-color components should be of equal bandwidth. This requires more storage space and bandwidth. In addition, processing an image in the RGB space is more complex since any change in the color of any pixel requires all the three RGB values to be read, calculation performed, and then stored. In the color information, is stored in the intensity and color format, some of the processing steps can be made faster.

As a result, Cb and Cr provide the hue[3] and saturation information of the color and Y' provides the brightness information of the color. Y' is defined to have a range of 16 to 235 and Cb and Cr have a range of 16 to 240 with 128 equal to zero (48).

Table 4-2 75% Y'CbCr Color Bars

	Normal Range	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
Y	16 to 235	180	162	131	112	84	65	35	16
Cb	16 to 240	128	44	156	72	184	100	212	128
Cr	16 to 240	128	142	44	58	198	212	114	128

[3] Hue is one of the tree main attributes of perceived color, in addition to lightness and chrome. Hue is also one of the tree dimensions in some color spaces along with saturation.

Figure 4-2 shows the Y'CbCr color cube (on the right) with the eight corners converted from the RGB color cube.

4.1.3 Converting from RGB to Y'CbCr

The main advantage of converting the image to the Y'CbCr domain is that influence of luminosity can be removed during our image processing. In the RGB domain, each component of the picture (red, green and blue) has a different brightness.

A color in the RGB color space is converted to the Y'CbCr color space using the following equations[4]:

$$Y' = 0.257R + 0.504G + 0.098B + 16$$
$$Cb = -0.148R - 0.291G + 0.439B + 128$$
$$Cr = 0.439R - 0.368G - 0.071B + 128$$

The gamma gamma-corrected RGB values, are represented as R'G'B'. However, in this case the input value of RGB is not gamma corrected.

A paper on skin color model in Y'CbCr color space for face detection (51), presents a plot with the tree axes showing the analysis to skin color pixels. The Cb and Cr components give a good indication on whether a pixel is a part of the skin or not. Another resent paper (21), plots the Cb and Cr components of the color space Y'CbCr and establish that is not necessary to plot the Y' component of luminosity.

4.1.4 Implementation for Skin Color Detection

For this thesis, I created a sample picture with skin color taken with the digital camera of the mobile phone (presented in section 3.2.5) to prevent

[4] The basic color space conversion equations are defined in ITU-R BT.601 standard

differences of the lens quality. The skin color sample is shown in Figure 4-3. These clips are taken from different lighting conditions and people from diverse ethnicity.

The goal of this sample is to identify the limits for skin detection in digital images taken with the mobile phone camera SE Z520i (presented in section 3.2.5).

Figure 4-3 Skin Color clips from different pictures to generate Cr Cb graphic

The skin color distribution can clearly be seen in Figure 4-4 and Figure 4-5, which are the Cb and Cr values of all the pixels that are part of the face in the image presented just above. These images are generated with Matlab (source code can be found in Appendix A). There is a strong correlation between the Cb and Cr component of the entire training image, to reveal the comparison between faces and non faces in the Y'CbCr space I also run the program with other images containing not just skin color, but background and non skin colors; and the results are shown in Figure 4-6. From this figure, it is apparent that background and faces can be distinguished by applying maximum and minimum threshold values for both Cb and Cr components. Note, however, that even with proper thresholds, images containing other parts of the body, such as exposed arms, legs and other skin, will be captured, and most be removed in following image processing steps.

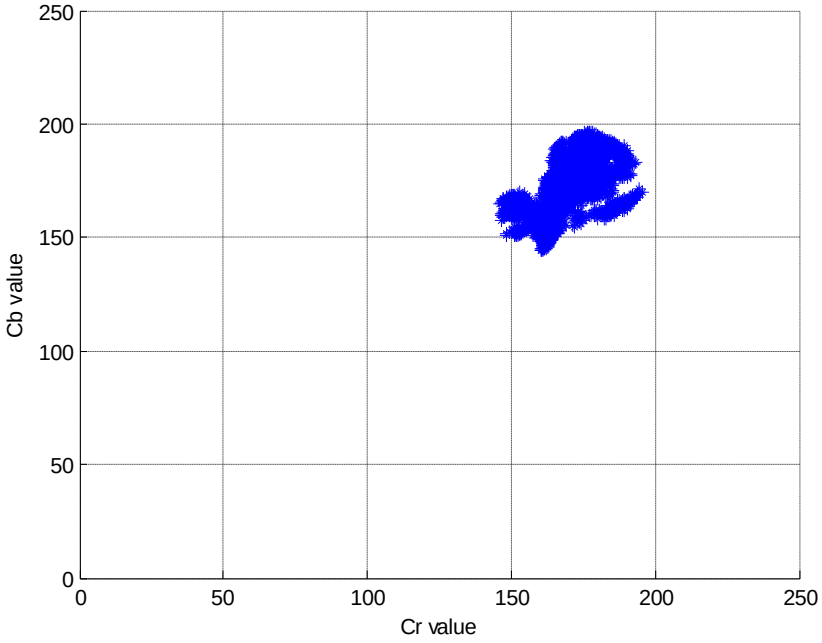

Figure 4-4 Skin pixel in Y′CbCr color space plotting Cr and Cb

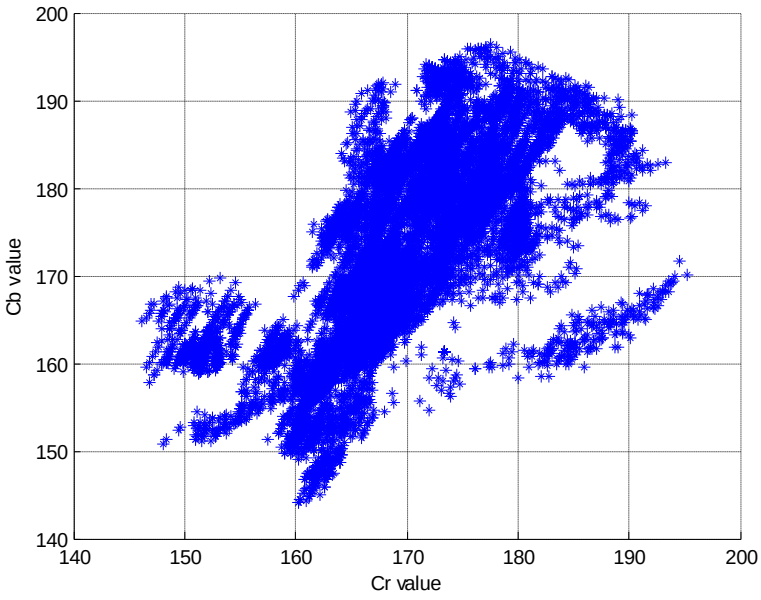

Figure 4-5 Skin pixel in Y′CbCr color space plotting Cr and Cb

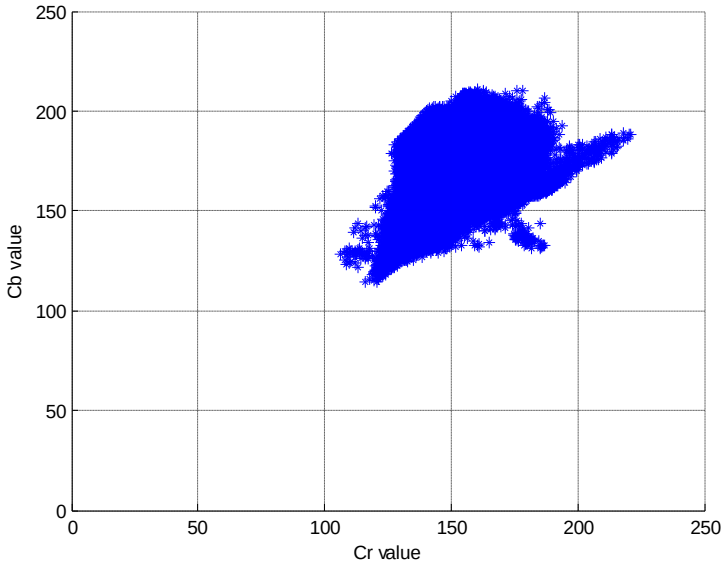

Figure 4-6 Y'CbCr transformation of the entire image

The thresholds where chosen based on the plot from Figure 4-5 and are as following:

$$145 < Cr < 196$$
$$150 < Cb < 190$$

Narrowing down these thresholds increases the probability that the accepted pixels are actually part of the skin and that most of the background is rejected. The main disadvantage is that also a lot of actual skin pixels are rejected, which might be a problem in future steps in the algorithm. Broadening the thresholds obviously detects most of the skin pixels, but will also include a lot of non-skin pixels. The values above gave us the best results.

An alternative to threshold in the Y'CbCr space is to use the HSV domain (color hues, saturations and values). Appendix B presents thresholds based on the hue and saturation values in a similar method to that used previously for the Y'CbCr space.

4.2 Application Design

For having a better overview and flexibility on the program, the Model-View-Controller architecture (MVC) is implemented. In everyday software development, is often encounter the same problem over and over again. Rather than solve the same problem each time, it makes sense to document the solution and re-use it. Design patterns are essentially documented, generalized solutions to common problems in application development. There are two phases for implementing a design pattern: understanding it then using it. In this section the MVC is explained.

4.2.1 What is MVC?

The MVC pattern consists of the three parts: model, view and controller. The *model* represents the data and rules that govern access to and updates of this data. The *view* is the actual screen that displays data and related commands to the user. The *controller* receives user actions and dispatches them to the model.

Figure 4-7 is a simple diagram of the MVC pattern. The view contains the graphical user interface, which is the part of the application the user can see. When the user alters the data on the screen and then initiates an action that action goes to the controller. The controller determines what kind of action has been requested and calls the appropriate interfaces of the model. The model can consist of several components, classes or packages, depending on the technology that implement the application's business logic.

Notice that the view has registered itself to the model. The model notifies the view of data changes in its various components. This way, the view does not have to poll the model about changes in the application data.

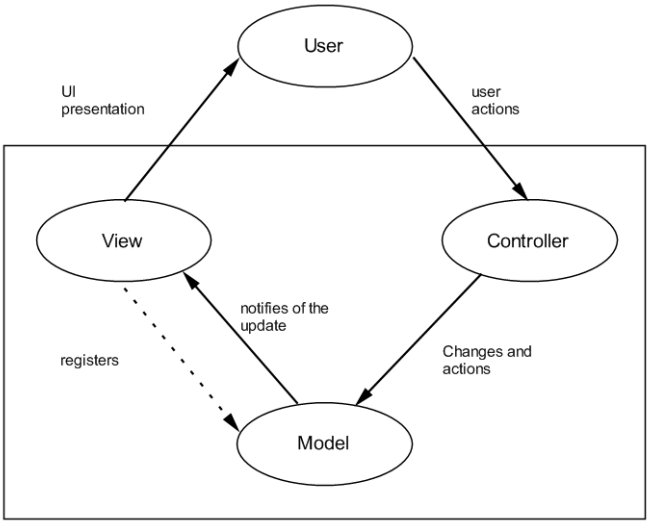

Figure 4-7 Elements of the MVC design pattern

Chapter 5

Implementation and System Integration

This chapter will give a description on the program that was developed for this thesis. Simplified code samples are provided throughout the chapter, aiming to show how the MIDlet is completed and the different approaches. Complete source code can be found in Appendix C. Section 5.1 describes the structure of the program, presenting the MVC model and showing the UML diagram of the general approach. Next, in section 5.2 a general description of the classes is presented. In section 5.3 the image analysis is explained, the implementation of the approaches include *black areas*, *motion* and *skin color areas analysis*. In section 5.4 a comparison of the implementation and experimental results are given. Finally a short description of the Robot Controller implementation is presented in section 5.5.

5.1 Structure of the program

5.1.1 MVC implementation

The MVC model design presented in section 4.2, is used for achieving a flexible and efficient structure. The classes are divided as shown in Figure 5-1.

The solid lines indicate a direct association, and the dashed lines indicate an indirect association.

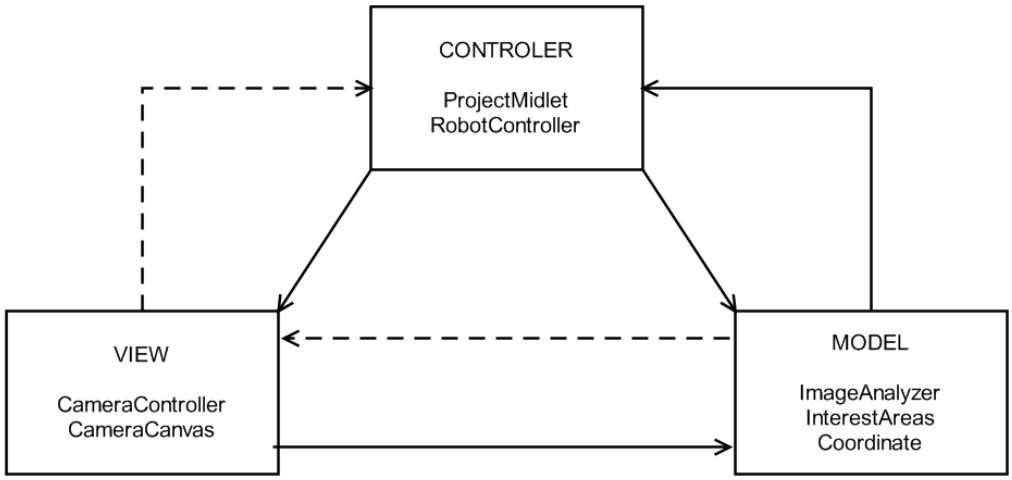

Figure 5-1 MVC implementation

In this implementation there are stationary classes, which are included in the Controller and View. The Model, on the other hand is changed according the approach. This design gives a wide flexibility for tuning to the needs of the program towards the different solutions.

To get a better explanation of the use of each class, section 5.2 will present a general description of the program.

5.1.2 UML diagram

The structure of the application implemented for this thesis, is shown in the UML diagram (Unified Modeling Language) of the program (Figure 5-2). The UML diagram contains the graphical representation of the classes and their connections. The name of the different classes and the methods implemented on

each of them are presented. The arrows point to the direct connection between the modules.

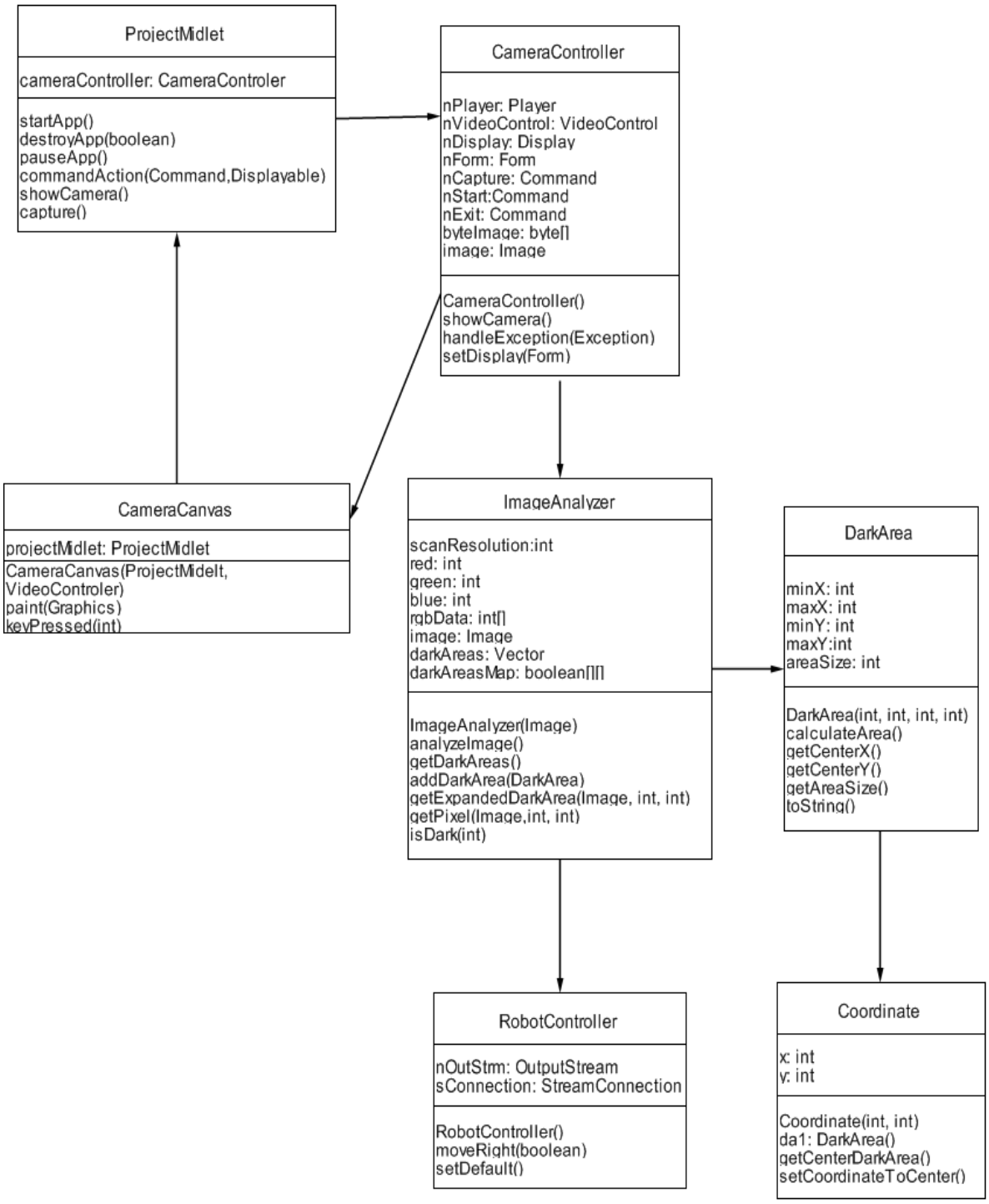

Figure 5-2 UML diagram

5.2 Class description

5.2.1 ProjectMIDlet

The ProjectMIDlet is the main controller of the program. As presented in section 3.2.4.1, this is the part of the program used to start, stop, pause and destroy the application. The ProjectMIDlet is the most important file within J2ME programming.

When the ProjectMIDlet is started, it generates all the necessary objects to run the program. This MIDlet creates the bridge between the CameraController and Camera Canvas. This part of the program also has an indirect feedback from the View by reading the key pushed by the user.

5.2.2 Graphical User Interface (GUI)

The GUI implementation contains two classes, CameraController and CameraCanvas. These classes are implemented in this project to control what will be shown on the display of the mobile phone. This is the interaction part between the application and the user.

CameraController contains the commands actions chosen by the user. The class CameraCanvas controls what is shown on the display. This of the GUI is necessary to be able to show online video of the camera. The CameraCanvas in general, controls what is shown in the display.

5.2.3 Image Processing

The Image Processing implementation is the most important part in this thesis. The ImageAnalyzer starts its functions when a picture is taken and sent to analyze. This class reads in the image and creates an object with the outer coordinates of the area of interest. For the reason that working with images is source consuming, from this point, it also generates a Boolean array with the same size as the image and map with "true" all the pixels that fulfill the requirements. Finally in a list of Vectors is saved the objects created. In section 5.3, the image analyzer is described.

5.2.4 Coordinate

The Coordinate will receive a vector of Objects created by the ImageAnalyzer. This class will compare the parameters established and decide which Object best accomplishes the requirements.

The best ranked Object or Objects, will generate a coordinate of its center. The Model section of the applications ends up when the coordinate is sent as (x, y) to the Robot Controller.

5.2.5 Robot controller

After the analysis of the picture is done, the coordinate is sent to the RobotController as (x, y).

This part of the program generates a connection with the external Bluetooth device blu2 and sends commands for the robot to move according with the position of the target.

Robot Controller gets the coordinate and generates the instructions to move it to the coordinate in the middle of the picture. In this case the entire

picture is 160x120 pixels. This will mean that the middle coordinate will have to be (80, 60)

The proposed implementation of the robot controller, uses Bluetooth communication and output stream. As shown in section 3.3.2.1, the development kit from EZURiO can be controlled with a R232 virtual port. And by using AT commands we can start and stop a signal on the circuit. The implementation is shown in section 5.5.

5.3 Image Analysis Implementation

The main focus of this thesis is the picture analysis or the Model part of the program (section 5.1) to achieve face detection on a mobile phone. A solution for reducing time during reading input data of the picture is presented by introducing the parameter *Scan Resolution* in section 5.3.1. The approaches to image analyzer I implemented and tested are:

- Black Areas (section 5.3.2)
- Motion (section 5.3.3)
- Skin color detection (section 5.3.4)

Finally, the results of the experiments are in section 5.4. The testes on the different image analysis are run with Eclipse and Wireless Toolkit from Sony Ericsson (3.2.6). Nevertheless, this tools use the mobile phone to run the analysis. That gives a better overview of the time it will take without the use of the utility tools.

5.3.1 Optimizing the Search

Reading the input image can be time consuming. During the early implementation of the algorithm, I found that when reading the entire image for analyzing, the program has to read $160 \times 120 = 19200\ pixels$.

With the reduced capacity of the mobile phone, the scan and analysis of the image takes in average 10 seconds to be completed. The need of implementing a method to enhance the speed gave as a result the creation of the parameter called *scan resolution*.

Scan resolution is defined by the steps between each checked pixel in the image. In Figure 5-3 an example is presented of the *scan resolution* equal 2 and 4. Where the dark painted areas are the checked pixels and the light painted areas are the not checked pixels.

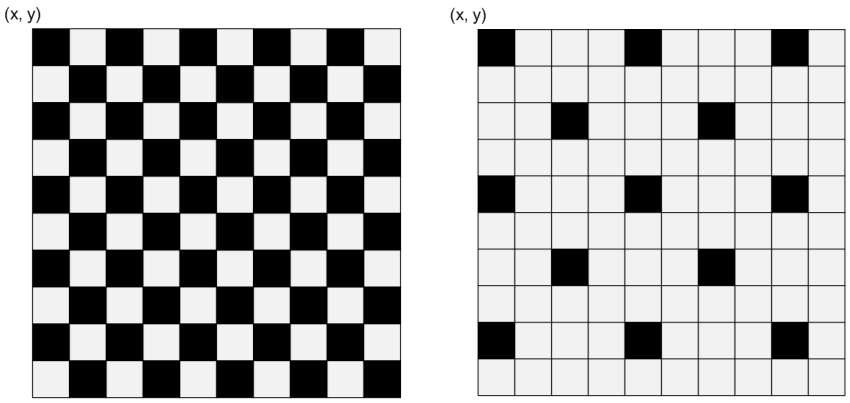

Figure 5-3 Pixel check with scan resolutions of 2 and 4

To compare the time and accuracy of the parameter *scan resolution*, it was created an example image with random dark elements in different sizes. The dark areas approach (section 5.3.2) was used as base for the testes. The same test picture was taken several times and run with the different scan resolution. The average time and results of accuracy are shown in Table 5-1.

Table 5-1 Evaluation of test results with different scan resolutions.

Resolution	Checked pixels	Average Time (s)	Accuracy (items found)	Reduced speed (%)	Comments
1 – All the pixels are checked	19200	10.52	100%	0%	All the elements are detected, is taking a long time
2 - each second pixel is checked	9600	6.18	100%	41.24%	All the elements are detected, but it still uses a long time to check the entire image
4 – every forth pixel	2400	3.27	83%	68.86%	Just elements smaller than 3x3 pixels are sometimes not detected. Speed improved but accuracy reduced
6 - every sixth pixel	1080	2.58	50%	75.50%	Elements of 5x5 are sometimes not detected, the improvement of the runtime drops
8 – every 8th pixel is checked	600	2.22	40%	78.90%	Elements of 7x7 are not detected, the improvement of the time is almost undetectable

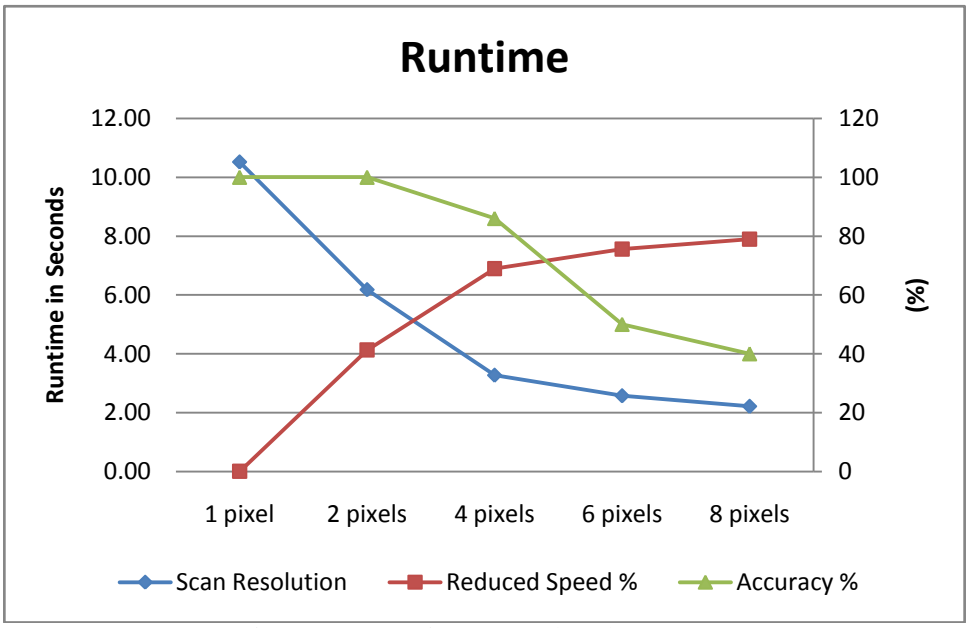

Figure 5-4 Runtime: Speed and Accuracy

After analyzing Table 5-1 and comparing the data with help of Figure 5-4, I propose to use the *scan resolution* of 4. This resolution gives both speed improvement and accuracy. The time improves by almost 70% and the accuracy still high in 83% of the cases. The implementation of the parameter scan resolution is shown in the code below.

```
for(int x=0; x<image.getWidth(); x+=scanResolution/2){
        if (isOdd){
                initialValue = scanResolution/2;
                isOdd = false;
        } else {
                initialValue = 0;
                isOdd = true;
        }
        for(int y=initialValue;y<image.getHeight();y+=scanResolution){
                        //This implementation reads the pixel according with
                        //the resolution.
        }
}
```

5.3.2 Dark Areas Analysis

The first approach is defined as localization of dark areas on a white background. This is a simplified approach for image processing. The analysis

first gets a picture and read the pixels according with the scan resolution defined. To identify a dark pixel and prevent noise, a threshold of luminance is established. By using the formula of Y' introduced in section 4.1.3 the luminance is calculated.

$$Y' = 0.257\,R + 0.504\,G + 0.098\,B + 16$$
$$Dark\ pixel\ defined\ as\ Y' < 70$$

All pixels with luminance less than 70 are considered as Dark Pixels. The code used for this implementation is shown bellow.

```
public boolean isDark(int pixelValue){

      red   = (pixelValue & 0x00FF0000) >> 16;
      green = (pixelValue & 0x0000FF00) >> 8;
      blue  =  pixelValue & 0x000000FF;

double lum = (0.257*red)+(0.504*green)+(0.098*blue)+16;

      return lum < 70;
}
```

5.3.2.1 Implementation of Search for Dark Areas

As explained in section 5.3.1, the program do not check all the pixels. When searching for dark pixels and when the program finds a group of pixels that fulfill the requirement, there is the need to connect these pixels to one or two areas. In Figure 5-5 it is presented a fragment of an image that locates the same amount of dark pixels but presets two cases. The pixels found are either belonging to one single dark area or they are divided in to two different dark areas.

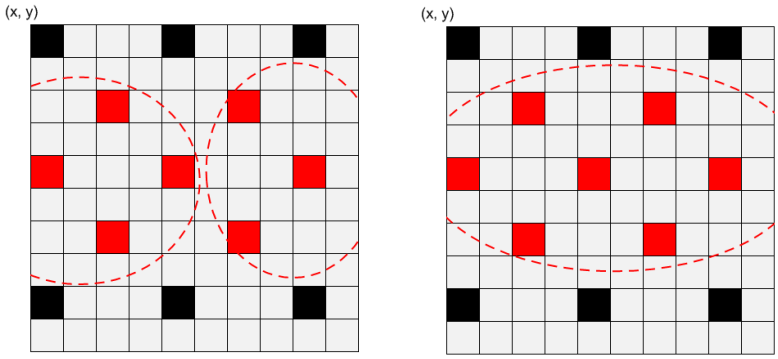

Figure 5-5 Cases of search

To connect each of these pixels to a dark area, the implemented algorithm first finds a dark pixel, and after will check the neighbor pixels until locate the entire object.

In Figure 5-6, the graphical representation of the search is shown. By assuming that a dark pixel is found, the search will expand to the upper and downer edge and define the minimum and maximum value of y. After it will move to the left and right to find the minimum and maximum value of x.

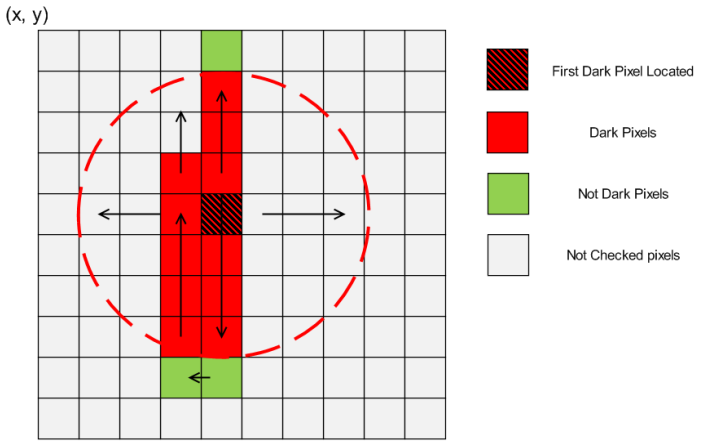

Figure 5-6 Searching in neighbor pixels

When the search is completed, the ImageAnalyzer class will create a DarkArea object with the coordinates of the outer edges of that area [(minX, minY), (maxX, maxY)].

5.3.2.2 One Dark Area

The first case of implementation within DarkAreas approach is to find one single dark area on a white background (Figure 5-7). With the implementation shown above, it can be found any dark area within any background since luminance is the only parameter used.

Figure 5-7 One dark area in figure

The Coordinate class will give back the (x, y) values of the center of the dark area.

5.3.2.3 Two dark areas

The second case for DarkAreas implementation is a simulation of the eyes of a person. In this case the algorithm will give back the coordinate of the middle point between the two areas.

For this implementation and to avoid noise, once found all the dark Areas and creating the list with all this elements found; this list is sent to Coordinate class to analyze each area to find two which are the same size.

Figure 5-8 Two dark areas in figure

In a real world application, looking for the eyeball or eyebrow of a person is an option. However, those two areas are not symmetrical when the person is turning to the side or when the lighting conditions changes. In addition the algorithm could also find dark spots by shadow of mouth or nose. Therefore, the algorithm measures the size of each dark object located and compare it to the other ones. When we find match greater than 80% between the two dark objects, the coordinate created will be the center of those two objects. For example in Figure 5-9 is presented that by searching for the dark areas in the image, such as eyebrow, mustache and beard a face could be found. The coordinate the implementation will give is the one between the eyebrows.

Figure 5-9 Finding dark areas in face detection application

5.3.3 Motion Analysis

Motion analysis was introduced in section 2.2.1.4. This algorithm takes as reference two pictures or a sequence of pictures and checks for differences. By finding image displacement, it can be calculated where the face of the person is located.

Assuming we have two successive images of a person with a minimum change as shown in Figure 5-10, it can be calculated the differences and identify the edge of the figure.

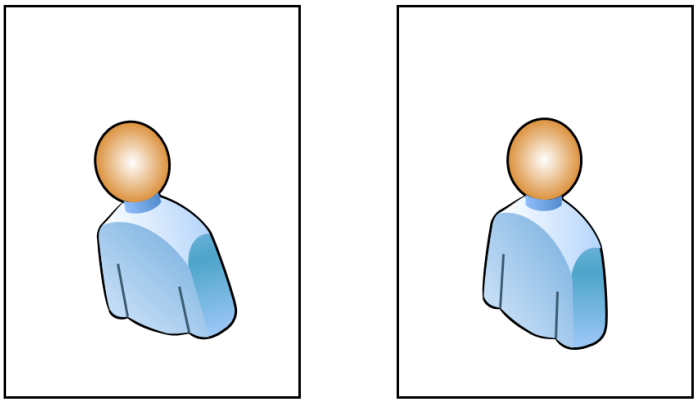

Figure 5-10 Motion implementation

In Figure 5-11 the edges of the person are marked with dashed line. The most of the situations the person taken in the picture will either move the *head and torso*, or the *entire body*. To get a close approach to the location of the face, the coordinate is calculated by the following equations:

$$x = \frac{Edge\ Width}{2}$$

$$y = \frac{Edge\ Height}{3}$$

The x-*coordinate* is given by the half of the moved image and y-*coordinate* by one third from the top. This *coordinate* is more likely to be closer to the face of the tester.

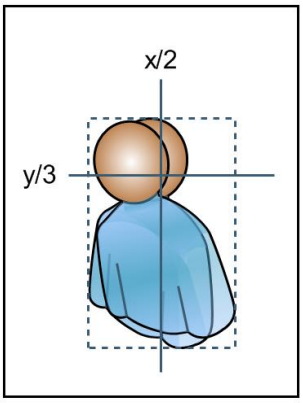

Figure 5-11 Comparing two images

The processing time is decreased since there is no value calculation for each pixel, we just compare them. However, the accuracy of finding faces can be reduced since other features are not relevant.

When implementing to the robot, the algorithms will take two successive images and analyze them. After it will enter to a waiting mode, calculated to the time the robot will use to turn and then take two other images. This prevents the motion of the robot to interfere with the approach.

5.3.4 Skin Color Analysis

Skin color algorithm (introduced section 2.2.1.1) is tested with the Y'CbCr color space (presented in section 4.1.2). The approach is defined as finding skin colored regions in a picture to identify faces.

As explained in section 4.1.3, Y'CbCr color space, need the RGB values of each pixel. With the function getRGB() the value of each element is extracted and used for the implementation. The implementation is shown below.

```
cb=(int) (0.148*red - 0.291*green + 0.439*blue + 128);
cr=(int) (0.439*red - 0.368*green + 0.071*blue + 128);

//new limits
if (145<cb && cb<189){
      if(145<cr && cr<196){
            return true;
      }else{
            return false;
      }
}else{
      return false;
}
```

To test the implementation it was used the skin sample from Figure 4-3. The limits did not change from the Matlab implementation shown in section 4.1.4. However to increase the accuracy the new limits defined are

$$145 < Cr < 196$$
$$145 < Cb < 189$$

I used the basis of the algorithm implemented for finding BlackAreas, with a modification for dropping some pixels inside the skin colored area when they are not skin color. As explained in section 5.3.2.1, the first thing to do is get a pixel that satisfies the criteria, in this case a skin colored pixel. After the neighbor pixels are checked, and as the implementation on black areas the limits of x and y are defined every time a new extreme value is found. The extreme

values are the edges of the skin colored area. The difference with the former approach is by searching the image as long as it is still inside the edges. If it finds a new edge, the extreme values will be redefined. The Figure 5-12 shows the new search method. This modification was done thus the eyes, noise or other shadows on the face can be excluded and the outer edge of the object can be found.

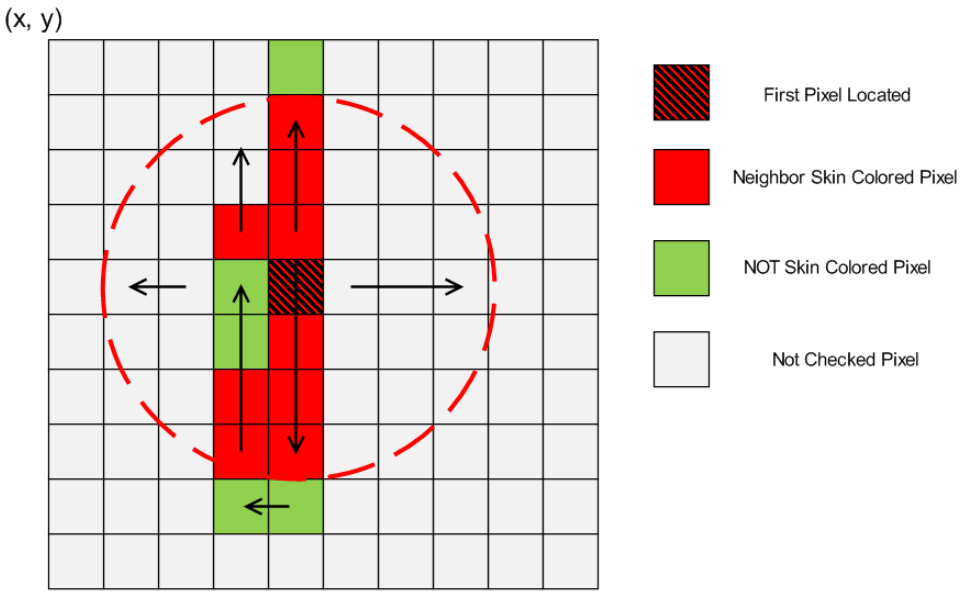

Figure 5-12 Modification of the original algorithm

The Coordinate class will give back the center coordinate to the biggest object found. This can either represent the biggest part of the face found or the closest face to the camera. By applying this solution the program dismisses other skin colored regions that can disturb the result, such as hands, arms and other not covered skin color regions.

5.3.4.1 Controlled Situation

The first testes where done under controlled situations of the picture. In other words, background is white and the face alone in the picture.

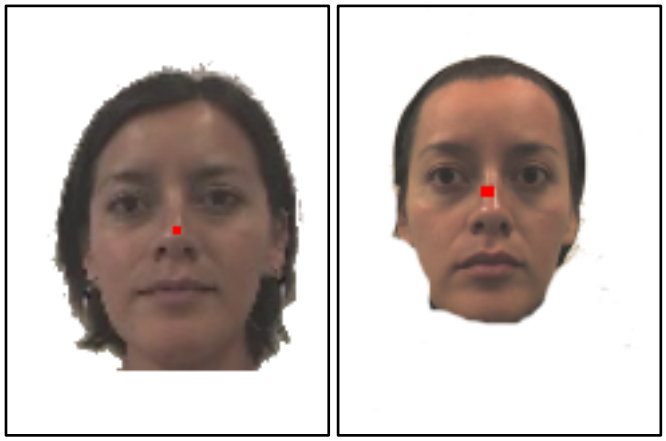

Figure 5-13 Test example

This first test was done for checking the ability of the algorithm to find the entire face dismissing shadows from nose, mouth and other areas as eyeball or eyebrow.

5.3.4.2 Not Controlled Situations

This section includes analysis of pictures in different backgrounds and other distracters for face analyzing.

5.3.4.2.1 Faces with glasses, mustache and beard

It is very common that the tester uses glasses, mustache or beard. Since the limits for chrominance are wide, the faces are found. In Figure 5-15 an example of face found in images with these distracters.

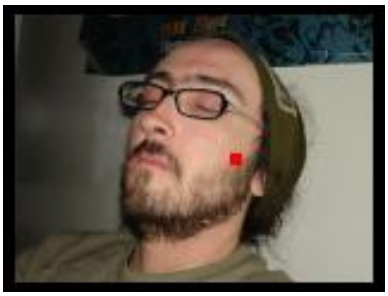

Figure 5-14 Image with glasses and beard

Figure 5-15 Faces with glasses

5.3.4.2.2 Skin colored areas close to faces

When skin colored areas are close to faces, the algorithm takes the entire area as one. In Figure 5-16 is shown that the located coordinate indicates the middle of the skin colored area. In the first and second picture from left it can be seen that their uncovered neck is read as a skin colored area. In the third picture, the hands close to the face makes the algorithm believe the face is longer and therefore sets the coordinate lower than to the center of the face.

Figure 5-16 Bigger skin colored area than face

In cases as the one in Figure 5-17 the two faces are consider as just one skin region. Therefore the middle coordinate is mist located.

Figure 5-17 Faces close to each other can be detected as one single person

5.3.4.2.3 Faces with disturbing lighting conditions

When faces are not looking directly to the camera, the shadows and angle modify the coordinate result. In the most of the cases the middle coordinate does not mean it will be the middle of the face.

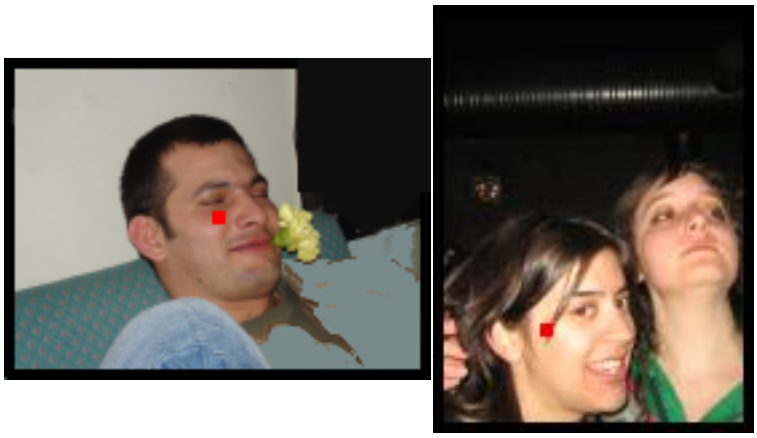

Figure 5-18 Images when the faces are not looking directly to the camera

In Figure 5-18 and Figure 5-19 the position and lighting condition disturb the result of the finding by not giving the center coordinate of the face. However, the face is found and the limitation is not a drawback since the face is still being found.

Figure 5-19 Images where the center is changed by lighting conditions

Shadows are another type of consideration generated by lighting conditions. When shadows around the eyes and nose can divide the face in different sections, the program just locates the biggest of these areas. In Figure 5-18 the cheek was the result of the search.

Figure 5-20 the lighting conditions let the algorithm just find one of the cheek

5.3.4.2.4 Redefine limits of Cb and Cr

In section 4.1.4, it was mentioned that by narrowing the limits the accuracy on skin detection could be increased. In some of the tests carried out during the implementation of the algorithm, I found some images that gave a

wrong result. However, the limits of Cb and Cr where redefined and in most of the cases this helped to increase accuracy.

The following pictures show both analyses, with former and present limits. The new Cb and Cr values are

$$155 < Cb < 185$$
$$155 < Cr < 180$$

In Figure 5-21, a part of the surrounding background was mistaken as skin colored. However when the limits were redefined, the face detection gave a better result.

Figure 5-21 Implementation with new limits of Cr and Cb

In Figure 5-22, the clothes were mistaken as skin colored area. As we can see in the picture to the right, this could be corrected by narrowing the limits of Cb and Cr.

Figure 5-22 Mismatch with clothes color, and new limits increase accuracy

The lighting condition of the case presented on Figure 5-23, lets almost all the pixels be detected as skin colored for their similarity to the color. However when narrowing the limits, the face is located correctly.

Figure 5-23 Error distinguishing skin color with background color

Nevertheless, not all the cases this modification is useful. In Figure 5-24 the surrounding yellow area has a similar value as the skin color. Therefore on the picture to the left side the mismatch consider parts of the yellow area. However, the picture to the right does not locate the face, but the shoulder. This case, the shadow areas of the faces are dismissed and therefore not located.

Figure 5-24 Surrounding colors as yellow can affect the recognition

These values for Cb and Cr have proven to be more efficient than the first proposed. However the test bench used for this thesis was adequate to test the different situations a picture analysis can be done.

5.4 Comparing Experimental Results

Localization rates across different approaches are presented in Table 5-2. With these results I can say that probably the motion algorithm is the most efficient for this situation. However, the parameters of the motion algorithm are not created for finding faces but just motion elements.

The goal is to locate faces so combining the three methods could be a good solution. First identifying the motion area, and inside that area skin color areas. After that looking for dark areas on the skin colored areas to find features as eyes nose and mouth. That combination of methods may give a big improvement of the results.

Table 5-2 Benchmarking of System Implementation

Type	Hits	Misses	False positive	Used time
One darkArea	99%	1%	The dark area is found even with other background than white. Just elements smaller than the scan resolution step are not found.	3.5 sec
Two darkAreas	97%	3%	Sometimes mismatch of comparison of the two areas.	3.5 sec
Motion	100%	0%	When there are no faces in motion area it will give a false positive. But since the goal of the approach is just finding the edges of the object in action, the hits are 100%	2.3 sec
SkinColor	87%	13%	When lighting conditions interfere in the search, the false positive is higher. However, in section 5.3.4.2.4 I narrowed the limits of Cr and Cb and that gave a better result on the implementation	10.9 sec

5.5 Robot Controller Implementation

Along the development of the project we had some problems connecting with the EZURiO antenna. The RobotController class was created to generate commands to be sent from the mobile phone via Bluetooth to the robot. However, the command was not tested due to the failure of the hardware.

Once the first version of the image analyzer was tested, we implement the connection with the robot. As explained along the project, the analyzer generates a coordinate which gives the idea of where face is. This coordinate is sent to RobotController and opens a connection with the antenna from EZURiO.

In the beginning we were able to communicate with the hardware, but after some tests, the antenna blocked the connection and did not let us connect again. We deliberate that may be configured to be hidden. That is one of the features of Bluetooth. However, we did not manage to contact via serial port either. When the second antenna undergoes the same problem, we started searching for documentation about the problem.

The RobotController class is implemented but not tested. It gets the coordinate from the ImageAnalyzer and calculates the difference between the coordinate and the middle of the picture. To create a Bluetooth connection we use the code shown bellow.

```
public RobotController() {
   try {
        sConnection = (StreamConnection)

        Connector.open("btspp://0080989897d8:1;authenticate=true;encrypt=false;mas
      ter=false");
        // 0080989897d8 is the address of Blu2i from Ezurio

        nOutStrm = sConnection.openOutputStream();

        // writes "esc" !!!<cr> for Blu2i to go on AT command mode
        try {Thread.sleep(1100);} catch (InterruptedException e){}
        nOutStrm.write(33); // !
        try {Thread.sleep(1100);} catch (InterruptedException e){}
        nOutStrm.write(33); // !
        try {Thread.sleep(1100);} catch (InterruptedException e){}
        nOutStrm.write(33); // !
        try {Thread.sleep(1100);} catch (InterruptedException e){}
        nOutStrm.write(13); // <cr>
        try {Thread.sleep(200);} catch (InterruptedException e){}
        }
        catch ( Exception e) {}

   }
```

This code will open a stream connection to the antenna that can simulate serial connection protocol. AT commands can used for sending the signal to the hardware. In theory, the connection could be done and the robot will not take long to interact with the faces around its environment.

Chapter 6

Conclusion and Proposal for Further Work

6.1 Conclusion

A face detection algorithm that uses several features in the image to perform detection is proposed. The algorithm is able to detect skin colored areas or areas with certain luminosity and in this way try to locate the (x, y) coordinate of the center of the face. This coordinate is ready to be sent to the blu2 antenna to direct the robot towards the face. This may simulate a human-robot interaction and create the *face-to-face* contact between robots and humans.

The work done in this master thesis is divided into two parts. The first part and biggest is the image processing. The search space is reduced by 87.5% with the implementation of the parameter *scan resolution*. The time is improved by 68.86%. The image analysis for face detection has an accuracy of 87%. The second part of the thesis is the implementation of the connection to a mobile robot. Different statements on robotics and HRI are given in the thesis, but the connection between the algorithm and the machinery was problematic due to

the delay of production of the second version of the robot and the lack of documentation from EZURiO.

The work on this project has been challenging, but ultimately extremely educational and rewarding. I feel I have had many programming challenges, but in addition to the purely technical, topics such as face detection, Bluetooth communication and the troubles we had with the EZURiO antenna, have also been personal satisfying. Face detection has been a subject that not only I have found fascinating, but that has engaged others, both within and outside of the field of computer science.

6.2 Discussion

During the development of this thesis, I have found different methods of improving the search. One of the limitations I had was the limited resources to work within the mobile phone.

An overview of the J2ME technology has been included so that readers with little or no knowledge of J2ME could understand the architecture and investigate the technology further on their own. References were included to J2ME resources, suitable as starting points for further study.

All of the code was written from scratch. This was largely motivated by a desire to have complete control in making the system as efficient as possible. Code written in J2ME is limited to the implementation specified in the API's of Sun (38). The MIDlet structure cannot be changed and the configurations for the mobile phone did not let a bigger resolution of the picture. Consequently, a lot of time was used designing, implementing and debugging the various components. Once a working system had been implemented, further time was spent on optimizing the code.

I would have chosen to get a course in Digital Image Processing before this project was started. In that way, I could understand concepts easier than what I did earlier in this project.

6.3 Proposal for further work

The combination of the three algorithms presented in this thesis will be the first proposal work I would implement in future work. Inspired by the simple implementation of algorithms to detect faces on images, I would like to pursue the task of using neural networks or Genetic Algorithms for better results. I believe that the feedback would yield more efficient results than those quantified in this thesis. The focus should be on creating algorithms that enhance the performance of the search.

There are some things that limited the performance of this project. The main was the capacity of the mobile phone and its camera. Another thing is the speed. I proposed an algorithm that is fast and at the same time compact. However, a mobile phone with better capacity could have let us explore more complicated solutions to increase the efficiency of the implementation. Nevertheless, the mobile phone is rapidly developing its capacity and function. I think in the not so far future the mobile phone may include the use of cutting edge technology as better cameras and higher capacities.

Mobile phones are becoming the primary personal gateway to access information Face recognition on portable mobile phones may help to protect sensitive data. In a not so far future mobile phones may take over personal computers and credit cards. The need of creating a secure environment around them is then important.

Appendix A **Matlab application**

```
function skinColor(testFile)
%reads the image
a=imread(testFile);
%gets the size of the image
d=size(a);
%Gets RGB pr pixel
for x=1:d(1)
    for y=1:d(2)
        R=double (a(x,y,1));
        G=double (a(x,y,2));
        B=double (a(x,y,3));

        Y(x,y) =  0.257*R + 0.504*G + 0.098*B + 16;
        Cb(x,y)= 0.148*R - 0.291*G + 0.439*B + 128;
        Cr(x,y)= 0.439*R - 0.368*G + 0.071*B + 128;

    end
end

%figure();
axis([0 250 0 250]);
grid on;
hold on;
xlabel('Cr value');
ylabel('Cb value');
plot(Cr,Cb,'b*');
```

Appendix B **HSV values**

An alternative to threshold in the Y'CbCr space is to use the HSV domain (color hues, saturations and values). Here is presented threshold values based on the hue and saturation in a similar method to that used previously for the Y'CbCr space. The color values cannot be used due to their slowly varying distribution. When applying both the Y'CbCr and HSV technique, the same results were achieved, and thus the conclusion was drawn that there is no advantage in using both techniques in my image processing algorithm.

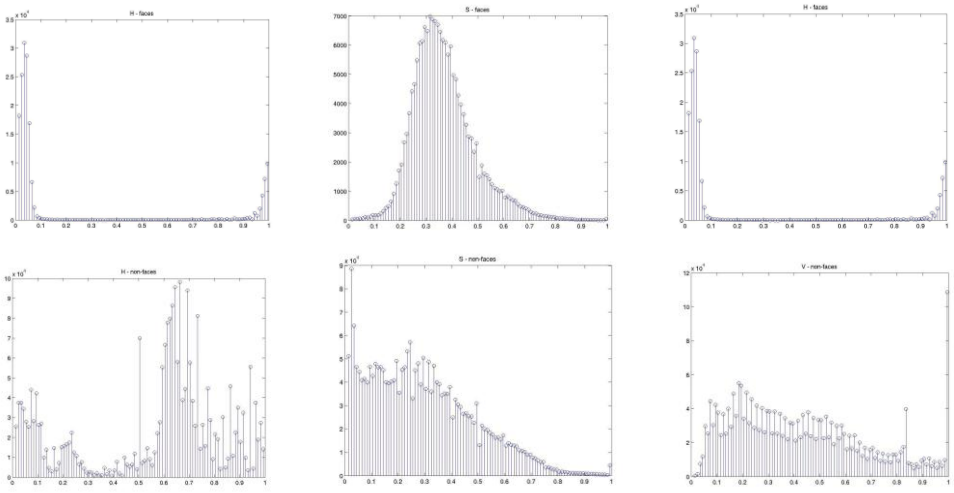

Figure B-1 Color Hue, Color Saturation and Color Values of the faces used for this experiment.

Appendix C Source Code

Parts of code that did not change in during implementation of the different approaches are presented at first.

ProjectMidlet.java

```
//import java.io.IOException;

import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
//import javax.microedition.lcdui.Image;
import javax.microedition.midlet.MIDlet;
import
javax.microedition.midlet.MIDletStateChangeException;

public class ProjectMidlet extends MIDlet implements
CommandListener{

  private CameraController cameraController;

  /*// For testing and checking results, this
implementation was made for reading files from
computer and analyzing
   * them with the phone ImageAnalyzer class
   protected void startApp() throws
MIDletStateChangeException {
    //For reading file from computer
    try {
      Image image = Image.createImage("test1_1.PNG");
      ImageAnalyzer imageAnalyzer = new
ImageAnalyzer(image);
      imageAnalyzer.analyzeImage();
      System.out.println("Founded areas: " +
imageAnalyzer.getSkinColorAreas());
      Coordinate coordinate = new
Coordinate(imageAnalyzer.getSkinColorAreas());
      destroyApp(true);
      notifyDestroyed();
      System.out.println("Coordinate" + coordinate);
    } catch(IOException e) {
      System.out.println(e.getMessage());
    }

  }*/

  //this method starts the application controling the
display
  protected void startApp()  throws
MIDletStateChangeException {
    cameraController = new CameraController();
    cameraController.nDisplay =
Display.getDisplay(this);
    String supports =
System.getProperty("video.snapshot.encodings");

    if(supports != null && supports.length()>0){
      cameraController.nForm.append("This aplication is
a part of my Master thesis."
          + "\n" + "Creating a system for eye
recognition and tracking");

  cameraController.nForm.addCommand(cameraController.n
Start);
    }else{
      cameraController.nForm.append("I'm sorry, this
device is not supported");
    }

    cameraController.nForm.setCommandListener(this);

  cameraController.setDisplay(cameraController.nForm);
  }

  //to destroy application
  protected void destroyApp(final boolean
unconditional){}

  //to pause application
  protected void pauseApp() {}

  //to read the commands selected on the phone
  public void commandAction(final Command c, final
Displayable s) {
    if(c.getCommandType() == Command.EXIT){
      destroyApp(true);
      notifyDestroyed();
    }else if (c.getLabel() == "Start"){
      showCamera();

    }else if (c.getLabel()== "Capture"){
      capture();
    }
  }

  //shows the camera picture online on the display
  public void showCamera() {
    cameraController.showCamera();
    Canvas canvas = new CameraCanvas (this,
cameraController.nVideoControl);
    canvas.addCommand(cameraController.nCapture);
    canvas.addCommand(cameraController.nExit);
    canvas.setCommandListener(this);
    cameraController.nDisplay.setCurrent(canvas);
  }

  //when buttom capture is pushed
  public void capture() {
    cameraController.capture();
  }
}
```

CameraController.java

```
import java.io.IOException;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.StringItem;
import javax.microedition.media.Manager;
import javax.microedition.media.MediaException;
```

```
import javax.microedition.media.Player;
import javax.microedition.media.control.VideoControl;


public class CameraController{

  public Player nPlayer;
  public VideoControl nVideoControl;
  public Display nDisplay;
  public Form nForm;
  public Command nCapture, nStart, nExit;

  private byte[] byteImage;
  private Image image;


  public CameraController(){
    nForm = new Form("testing");
    nExit = new Command("Exit", Command.EXIT, 0);
    nStart = new Command("Start", Command.SCREEN,0);
    nCapture = new Command
("Capture",Command.SCREEN,0);

    nForm.addCommand(nExit);
  }

  public void showCamera()  {
    try {
      nPlayer =
Manager.createPlayer("capture://video");
      nPlayer.realize();

      nVideoControl =
(VideoControl)nPlayer.getControl("VideoControl");
      nPlayer.start();
    }
    catch (IOException ioe) { handleException(ioe); }
    catch (MediaException me) { handleException(me); }
  }

  public void capture() {
    try {
      // Get the image.
      byteImage = nVideoControl.getSnapshot(null);

      //creating image of 160x120 pixels
      image = Image.createImage(byteImage, 0,
byteImage.length);

      ImageAnalyzer imageAnalyzer = new
ImageAnalyzer(image);
      imageAnalyzer.analyzeImage();
      //System.out.println("Dark Areas:\n" +
imageAnalyzer.getSkinColorAreas());

      if (nForm.size() > 0 && nForm.get(0) instanceof
StringItem){
        nForm.delete(0);
      }

      nDisplay.setCurrent(nForm);

      // Shut down the player.
      nPlayer.close();
      nPlayer = null;
      nVideoControl = null;
    }
    catch (MediaException me) { handleException(me); }
  }


  private void handleException(Exception e) {
    Alert a = new Alert("Exception", e.toString(),
null, null);
    a.setTimeout(Alert.FOREVER);
    nDisplay.setCurrent(a, nForm);
  }

  public void setDisplay(Form form){
    nDisplay.setCurrent(form);
  }
}
```

CameraCanvas.java

```
import javax.microedition.lcdui.*;
import javax.microedition.media.MediaException;
import javax.microedition.media.control.VideoControl;

public class CameraCanvas extends Canvas {
  private ProjectMidlet projectMidlet;

  public CameraCanvas(ProjectMidlet midlet,
VideoControl videoControl) {
    int width = getWidth();
    int height = getHeight();

    projectMidlet = midlet;


  videoControl.initDisplayMode(VideoControl.USE_DIRECT
_VIDEO, this);
    try {
      videoControl.setDisplayLocation(2, 2);
      videoControl.setDisplaySize(width - 4, height -
4);
    }
    catch (MediaException me) {
      try { videoControl.setDisplayFullScreen(true); }
      catch (MediaException me2) {}
    }
    videoControl.setVisible(true);
  }

  public void paint(Graphics g) {
    int width = getWidth();
    int height = getHeight();

    // Draw a green border around the VideoControl.
    g.setColor(0x0000ff);
    g.drawRect(0, 0, width - 1, height - 1);
    g.drawRect(1, 1, width - 3, height - 3);
  }

  public void keyPressed(int keyCode) {
    int action = getGameAction(keyCode);
    if (action == FIRE)
      projectMidlet.capture();
  }
}
```

RobotController.java

```
/* This class connects via bluetooth to the EZURiO
antenna
 * The antenna has id: 0080989897d8
 * Its connected via StreamConnection and uses AT
commands
 * to control the development kit
 */
import java.io.IOException;
import java.io.OutputStream;

import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;


public class RobotController {
  public OutputStream nOutStrm = null;
  public StreamConnection sConnection = null;

  public RobotController() {
    try {
      sConnection = (StreamConnection)

  Connector.open("btspp://0080989897d8:1;authenticate=
true;encrypt=false;master=false");
      // 0080989897d8 is the address of Blu2i from
Ezurio

      nOutStrm = sConnection.openOutputStream();

      // writes "esc" !!!<cr> for Blu2i to go on AT
command mode
      try {Thread.sleep(1100);} catch
(InterruptedException e){}
      nOutStrm.write(33); // !
      try {Thread.sleep(1100);} catch
(InterruptedException e){}
      nOutStrm.write(33); // !
      try {Thread.sleep(1100);} catch
(InterruptedException e){}
      nOutStrm.write(33); // !
      try {Thread.sleep(1100);} catch
(InterruptedException e){}
      nOutStrm.write(13); // <cr>
      try {Thread.sleep(200);} catch
(InterruptedException e){}
    }
    catch ( Exception e) {}
  }

  //TODO will get object Coorinate() and set the
position required

  public void moveRight(boolean goRight) {
    try {
      if (goRight) { // ats624=1 <cr>
        nOutStrm.write(65); // A
        nOutStrm.write(84); // T
        nOutStrm.write(83); // S
        nOutStrm.write(54); // 6
        nOutStrm.write(50); // 2
        nOutStrm.write(52); // 4
        nOutStrm.write(61); // =
        nOutStrm.write(49); // 1
        nOutStrm.write(13); // <cr>
      }
      else { // ats624=0 <cr>
        nOutStrm.write(65); // A
        nOutStrm.write(84); // T
        nOutStrm.write(83); // S
        nOutStrm.write(54); // 6
        nOutStrm.write(50); // 2
        nOutStrm.write(52); // 4
        nOutStrm.write(61); // =
        nOutStrm.write(48); // 0
        nOutStrm.write(13); // <cr>
      }
    }
    catch (IOException e) {}
  }

  public void setDefault(){
    try{
      //ATS621=0<cr>
      nOutStrm.write(65); // A
      nOutStrm.write(84); // T
      nOutStrm.write(83); // S
      nOutStrm.write(54); // 6
      nOutStrm.write(50); // 2
      nOutStrm.write(49); // 1
      nOutStrm.write(61); // =
      nOutStrm.write(48); // 0
      nOutStrm.write(13); // <cr>

      //ATS622=0<cr>
      nOutStrm.write(65); // A
      nOutStrm.write(84); // T
      nOutStrm.write(83); // S
      nOutStrm.write(54); // 6
      nOutStrm.write(50); // 2
      nOutStrm.write(50); // 2
      nOutStrm.write(61); // =
      nOutStrm.write(48); // 0
      nOutStrm.write(13); // <cr>

      //ATS624=0<cr>
      nOutStrm.write(65); // A
      nOutStrm.write(84); // T
      nOutStrm.write(83); // S
      nOutStrm.write(54); // 6
      nOutStrm.write(50); // 2
      nOutStrm.write(52); // 4
      nOutStrm.write(61); // =
      nOutStrm.write(48); // 0
      nOutStrm.write(13); // <cr>

      //ATS625=0<cr>
      nOutStrm.write(65); // A
      nOutStrm.write(84); // T
      nOutStrm.write(83); // S
      nOutStrm.write(54); // 6
      nOutStrm.write(50); // 2
      nOutStrm.write(53); // 5
      nOutStrm.write(61); // =
      nOutStrm.write(48); // 0
      nOutStrm.write(13); // <cr>

    }
    catch (IOException e) {}
  }
}
```

The areas of the code that changed with each implementation are the following tree classes.

"Dark areas" approach. Section 5.3.2

ImageAnalyzer.java

```
/* This analyzer will get a image and give a x, y
coordinate of the middle of the 2 darkest point of the
 * picture. This will represent the eyes of a person
in the second version
 */


import java.util.Vector;

import javax.microedition.lcdui.Image;


public class ImageAnalyzer {

  private int scanResolution = 4;
  private int red, green, blue;
  private int[] rgbData;
  private Image image;
  private Vector darkAreas;
  private boolean[][] darkAreasMap;

  public int minX, maxX, minX2;
  public int minY, maxY;
  public int midX, midY;


  public ImageAnalyzer(Image image){
    this.image = image;
    darkAreas = new Vector();
    darkAreasMap = new
boolean[image.getWidth()][image.getHeight()];
  }

  public void analyzeImage(){
    boolean isOdd = false;
    int initialValue = 0;
    rgbData = new int[1];

    // i do a brief scan of image first to find dark
spots, and then investige each spot
    for(int x = 0; x < image.getWidth();
x+=scanResolution/2){
      if (isOdd){
        initialValue = scanResolution/2;
        isOdd = false;
      } else {
        initialValue = 0;
        isOdd = true;
      }
      for(int y = initialValue; y < image.getHeight();
y+=scanResolution){
        if(isDark(getPixel(image, x, y))){
          // i check if pixel is included in existing
dark area
          if(!darkAreasMap[x][y]){
            DarkArea darkArea =
getExpandedDarkArea(image, x, y);
            addDarkArea(darkArea);
          }else{
            System.out.println("Was in map");
          }
        }
      }
    }
  }
```

```
  public Vector getDarkAreas() {
    return darkAreas;
  }

  public void addDarkArea(DarkArea darkArea) {
    darkAreas.addElement(darkArea);
    for(int x=darkArea.getMinX();
x<=darkArea.getMaxX();x++){
      for(int
y=darkArea.getMinY();y<=darkArea.getMaxY();y++){
        darkAreasMap[x][y]=true;
      }
    }
  }

  public DarkArea getExpandedDarkArea(Image image, int
x, int y){
    minX = x;
    minY = y;
    maxX = x;
    maxY = y;

    int currentX = x;
    int currentY = y;
    boolean moreToLeft = true;
    boolean moreToRight = true;
    int lastMinY = y;

    // i check everything left to start point
    while(moreToLeft){

      // i check if the starting pixel is black or
white
      if(isDark(getPixel(image, currentX, currentY))) {
        // i go up to first black on this line
        while(isDark(getPixel(image, currentX,
currentY))&&0!=currentY){
          currentY -= 1;
        }
        if(currentY+1<minY){
          minY = currentY+1;
        }
        lastMinY = currentY+1;
      } else {
        // i go down to the first black (or maxY)
        while(!(isDark(getPixel(image, currentX,
currentY)))) {
          if(currentY > maxY +1){
            // There was no black pixels on this row
            moreToLeft = false;
            minX = currentX+1;
            break;
          }
          currentY += 1;
        }
        lastMinY = currentY;
      }

      if(moreToLeft){
        // i go down to last black on this line
        while(isDark(getPixel(image, currentX,
currentY))&&currentY!=image.getHeight()){
          currentY += 1;
        }
```

```
      if(currentY==image.getHeight()){
        maxY = currentY;
      }else if(currentY-1>maxY){
        maxY = currentY-1;
      }
      currentX -= 1;
      currentY = minY;
    }
    currentY = lastMinY -1;
  }

  // i check everything right to start point
  currentX = x;
  currentY = y;
  while(moreToRight){
    // I check if the starting pixel is black or
white
    if(isDark(getPixel(image, currentX, currentY))) {
      // I go down to first black on this line
      while(isDark(getPixel(image, currentX,
currentY))&&0<=currentY &&
currentY<=image.getHeight()){
        currentY -= 1;
      }
      if(currentY<minY){
        minY = currentY;
      }
      lastMinY = currentY;
    } else {
      // I go up to the first black (or maxY)
      while(!(isDark(getPixel(image, currentX,
currentY)))) {
        if(currentY > maxY +1){
          // There was no black pixels on this row
          moreToRight = false;
          maxX = currentX-1;
          break;
        }
        currentY += 1;
      }
      lastMinY = currentY;
    }

    if(moreToRight){
      // I go up to last black on this line
      while(isDark(getPixel(image, currentX,
currentY))&& currentY<=image.getHeight()){
        currentY += 1;
      }
      if(currentY>maxY){
        maxY = currentY;
      }
      currentX += 1;
      currentY = minY;
    }
    currentY = lastMinY -1;
  }

  return new DarkArea(minX, maxX, minY, maxY);
 }

 public int getPixel(Image image, int x, int y){
  image.getRGB(rgbData, 0, 1, x, y, 1, 1);
  return rgbData[0];
 }

 public boolean isDark(int pixelValue){
  red   = (pixelValue & 0x00FF0000) >> 16;
  green = (pixelValue & 0x0000FF00) >> 8;
  blue  =  pixelValue & 0x000000FF;
  double lum = (red+green+blue)*0.33;
  return lum < 70;
 }
}
```

DarkArea.java

```
public class DarkArea {
 private int minX;
 private int maxX;
 private int minY;
 private int maxY;
 private int areaSize;

 public int centerX,centerY;

 public DarkArea(int minX, int maxX, int minY, int
maxY){
  this.minX = minX;
  this.maxX = maxX;
  this.minY = minY;
  this.maxY = maxY;
  calculateArea();
 }

 //calculates the area of the rectangular defined by
the 2 coordinates.
 //This is to be able to compare it
 public void calculateArea(){
  this.areaSize  =  (this.maxX-this.minX)*(this.maxY-
this.minY);
 }

 public int getCenterX(){
  this.centerX=(int) ((this.minX + this.maxX) * 0.5);
  return this.centerX;
 }

 public int getCenterY(){
  this.centerY=(int) ((this.minY + this.maxY) * 0.5);
  return this.centerY;
 }

 public int getMaxX() {
  return maxX;
 }

 public void setMaxX(int maxX) {
  this.maxX = maxX;
 }

 public int getMaxY() {
  return maxY;
 }

 public void setMaxY(int maxY) {
  this.maxY = maxY;
 }

 public int getMinX() {
  return minX;
 }

 public void setMinX(int minX) {
  this.minX = minX;
 }

 public int getMinY() {
  return minY;
 }

 public void setMinY(int minY) {
  this.minY = minY;
 }

 public int getAreaSize() {
  return areaSize;
 }

 public String toString(){
  return "Dark area: (" + minX + "," + minY + "), ("
+ maxX + "," + maxY + ")";
 }
}
```

Coordinate.java

```
/* Sets coordinate to the center point of area of
interest
 * If it is just one dark area on the picture, will
set coordinate to the
 * center of this one. If there is more than one,
will find the two which are
 * same size (with 80% thershold) and take the
center of those two
 *
 */

import java.util.Vector;

public class Coordinate {
  private int x;
  private int y;

  public DarkArea da1,da2;
  public int biggestSize, secondBiggest;

  public Coordinate(int x, int y){
    this.x=x;
    this.y=y;
  }

  public Coordinate (Vector darkAreas){
    int i = darkAreas.size();

    //it will give the center point of the only
object in the Vector.
    if(i==1){
      da1=(DarkArea)darkAreas.elementAt(0);
      this.x=da1.getCenterX();
      this.y=da1.getCenterY();

    //if is just 2 objects of type DarkArea will just
give back the center point of these two
    } else if(i==2){

  setCoordinateToCenterOfTheseTwoAreas((DarkArea)dar
kAreas.elementAt(0),
(DarkArea)darkAreas.elementAt(1));

    //if is more than 2 objects will choose the 2 who
are the same size in 80% accuracy and give the
center point
    } else {
      compareDarkAreas(darkAreas);
    }
  }

  //compares the areas and gives back the two which
are the same size.
  private void compareDarkAreas(Vector darkAreas) {
    double bestMatch=0;
    double percent;
    int numberOfElements = darkAreas.size();
    for(int i = 0; i<numberOfElements;i++){
      for(int j = 1; j<numberOfElements;j++){
        if(i<j){
          percent =
matchDarkAreaInPercentage((DarkArea)darkAreas.elemen
tAt(i),(DarkArea)darkAreas.elementAt(j));
          if(bestMatch < percent){
            bestMatch=percent;

  setCoordinateToCenterOfTheseTwoAreas((DarkArea)dar
kAreas.elementAt(i),
(DarkArea)darkAreas.elementAt(j));
          }
        }
      }
    }
  }

  // gives back the match percentage of the area
  private double matchDarkAreaInPercentage(DarkArea
area, DarkArea area2) {
    double matchValue;
    int areaS1 = area.getAreaSize();
    int areaS2 = area2.getAreaSize();
    if(areaS1<areaS2){
      matchValue=(double)areaS1/areaS2;
    }else{
      matchValue=(double)areaS2/areaS1;
    }
    return matchValue;
  }

  //Sets coordinate to the center of these two areas
  private void
setCoordinateToCenterOfTheseTwoAreas(DarkArea da1,
DarkArea da2) {
    this.x = (int) ((da1.getCenterX() +
da2.getCenterX())*0.5);
    this.y = (int) ((da1.getCenterY() +
da2.getCenterY())*0.5);
  }

  public String toString(){
    return "(" + x + ", "+ y +")";
  }

}
```

"Motion" approach. Section 5.3.3

ImageAnalyzer.java

```
/* This analyzer will get two images and compare the
pixels.
 * Will return the coordinate of the area that
contains motion
 */


import javax.microedition.lcdui.Image;


public class ImageAnalyzer {

  private int scanResolution = 4;
  private int[] rgbData;
  private Image image;
  private Image image2;
  private boolean[][] movedObjectMap;

  public int minX, maxX, minX2;
  public int minY, maxY;
  public int midX, midY;


  public ImageAnalyzer(Image image, Image image2){
    // I assume that both images are the same size.
    this.image = image;
    this.image2 = image2;

    movedObjectMap = new
boolean[image.getWidth()][image.getHeight()];
  }

  public void analyzeImage(){
    boolean isOdd = false;
    int initialValue = 0;
    rgbData = new int[1];
    minX=image.getWidth();
    minY=image.getHeight();
    maxX=0;
    maxY=0;
```

```
    // I do a brief scan of image first to find dark
spots, and then investige each spot
    for(int x = 0; x < image.getWidth();
x+=scanResolution/2){
      if (isOdd){
        initialValue = scanResolution/2;
        isOdd = false;
      } else {
        initialValue = 0;
        isOdd = true;
      }
      for(int y = initialValue; y < image.getHeight();
y+=scanResolution){

  if((getPixel(image,x,y)!=getPixel(image2,x,y))){
          if(x<minX){
            minX=x;
          }else{
            maxX=x;
          }
          if(y<minY){
            minY=y;
          }else if(maxY<y){
            maxY=y;
          }
          movedObjectMap[x][y]=true;
        }
      }
    }
    System.out.println("min max (" + minX + "," + minY
+ ") ("+maxX + "," +maxY +")");
    Coordinate coordinate = new Coordinate(minX, minY,
maxX,maxY);
    System.out.println(coordinate);
  }

  public int getPixel(Image image, int x, int y){
    image.getRGB(rgbData, 0, 1, x, y, 1, 1);
    return rgbData[0];
  }
}
```

MovedArea.java

```
public class MovedArea {
  private int minX;
  private int maxX;
  private int minY;
  private int maxY;
  private int areaSize;

  public int centerX,centerY;

  public MovedArea(int minX, int maxX, int minY, int
maxY){
    this.minX = minX;
    this.maxX = maxX;
    this.minY = minY;
    this.maxY = maxY;
    calculateArea();
  }

  //calculates the area of the rectangular defined by
the 2 coordinates. This is to be able to compare it
  public void calculateArea(){
    this.areaSize = (this.maxX-this.minX)*(this.maxY-
this.minY);
```

```
  }

  public int getCenterX(){
    this.centerX=(int) ((this.minX + this.maxX) * 0.5);
    return this.centerX;
  }

  public int getCenterY(){
    this.centerY=(int) ((this.minY + this.maxY) * 0.5);
    return this.centerY;
  }

  public int getMaxX() {
    return maxX;
  }

  public void setMaxX(int maxX) {
    this.maxX = maxX;
  }

  public int getMaxY() {
    return maxY;
  }
```

```
  public void setMaxY(int maxY) {
    this.maxY = maxY;
  }

  public int getMinX() {
    return minX;
  }

  public void setMinX(int minX) {
    this.minX = minX;
  }

  public int getMinY() {
    return minY;
```

```
  }

  public void setMinY(int minY) {
    this.minY = minY;
  }

  public int getAreaSize() {
    return areaSize;
  }

  public String toString(){
    return "Moved area: (" + minX + "," + minY + "), ("
+ maxX + "," + maxY + ")";
  }
}
```

Coordinate.java

```
//Sets coordinates to turn to the center in x and 1/3 from top of y


public class Coordinate {
  private int x;
  private int y;

  public Coordinate(int x, int y){
    this.x=x;
    this.y=y;
  }


  public Coordinate(int minX, int minY, int maxX, int maxY) {
    this.x=(int) ((minX+maxX)*.5);
    this.y=(int) ((minY+maxY)*.3); //to 1/3 of the top, it will be the eyes or face
  }


  public String toString(){
    return "(" + x + ", "+ y +")";
  }

}
```

"Skin Color" approach. Section 5.3.4

ImageAnalyzer.java

```
/* This analyzer will find the areas of color skin
 * create a object of type SkinColorArea for each of
these.
 * The skin color localization is done by using color
space CbCr
 * and using thersholding
 * 141<Cb<181
 * 138<Cr<181
 * Which gave better results. The biggest area will be
consider
 * The face closest to the camera and generates with
Coordinate class
 * the center coordinate of this area.
 */


import java.util.Vector;

import javax.microedition.lcdui.Image;


public class ImageAnalyzer {

  private int scanResolution = 4;
  private int red, green, blue;
  private int cb,cr;
  private int[] rgbData;
  private Image image;
  private Vector skinColorAreas;
  private boolean[][] skinColorAreasMap;

  public int minX, maxX, minX2;
  public int minY, maxY;
  public int midX, midY;

  int cbmin =1000;
  int cbmax;
  int crmin=1000;
  int crmax;
  int pixel=0;

  public ImageAnalyzer(Image image){
    this.image = image;
    skinColorAreas = new Vector();
    skinColorAreasMap = new
boolean[image.getWidth()][image.getHeight()];
  }

  public void analyzeImage(){
    boolean isOdd = false;
    int initialValue = 0;
    rgbData = new int[1];

    // I do a brief scan of image first to find skin
color spots, and then investige each spot
    for(int x = 0; x < image.getWidth();
x+=scanResolution/2){
      if (isOdd){
        initialValue = scanResolution/2;
        isOdd = false;
      } else {
        initialValue = 0;
        isOdd = true;
      }
      for(int y = initialValue; y < image.getHeight();
y+=scanResolution){
        pixel++;

        if(isSkinColor(getPixel(image, x, y))){
          // I check if pixel is included in existing
dark area
          if(!skinColorAreasMap[x][y]){
            SkinColorArea skinColorArea =
getExpandedSkinColorArea(image, x, y);
            addSkinColorArea(skinColorArea);
          }else{
            //System.out.println("Was in map");
          }
        }
      }
    }
    //System.out.println("Cb:(" + cbmin + ", "+cbmax+")
Crmin:(" + crmin+", "+crmax+")");
    //System.out.println("scan res:" + pixel);
  }

  public Vector getSkinColorAreas() {
    return skinColorAreas;
  }

  public void addSkinColorArea(SkinColorArea
skinColorArea) {
    skinColorAreas.addElement(skinColorArea);
    for(int x=skinColorArea.getMinX();
x<=skinColorArea.getMaxX();x++){
      for(int
y=skinColorArea.getMinY();y<=skinColorArea.getMaxY();y
++){
        skinColorAreasMap[x][y]=true;
      }
    }
  }


  public SkinColorArea getExpandedSkinColorArea(Image
image, int x, int y){
    minX = x;
    minY = y;
    maxX = x;
    maxY = y;

    int currentX = x;
    int currentY = y;
    boolean moreToLeft = true;
    boolean moreToRight = true;
    int lastMinY = y;

    // I check everything left to start point
    while(moreToLeft){

      // I check if the starting pixel is black or
white
      if(isSkinColor(getPixel(image, currentX,
currentY))) {
        // I go up to first black on this line
        while(isSkinColor(getPixel(image, currentX,
currentY))&&0!=currentY){
          if(currentY>0){
            currentY -= 1;
          }else{
            break;
          }
        }
        if(currentY+1<minY){
          minY = currentY+1;
        }
        lastMinY = currentY+1;
      } else {
        // I go down to the first black (or maxY)
        while(!(isSkinColor(getPixel(image, currentX,
currentY)))) {
          if(currentY > maxY +1){
            // There was no black pixels on this row
            moreToLeft = false;
            minX = currentX+1;
            break;
          }
          if(currentY<image.getHeight()){
            currentY += 1;
          }else{
            break;
          }
        }
        lastMinY = currentY;
```

```
      }

      if(moreToLeft){
        // I go down to last black on this line
        while(isSkinColor(getPixel(image, currentX,
currentY))&&currentY!=image.getHeight()){
          currentY += 1;
        }
        if(currentY==image.getHeight()){
          maxY = currentY;
        }else if(currentY-1>maxY){
          maxY = currentY-1;
        }
        currentX -= 1;
        currentY = minY;
      }
      currentY = lastMinY -1;
    }

    // I check everything right to start point
    currentX = x;
    currentY = y;
    while(moreToRight){
      // I check if the starting pixel is black or
white
      if(isSkinColor(getPixel(image, currentX,
currentY))) {
        // I go down to first black on this line
        while(isSkinColor(getPixel(image, currentX,
currentY))&&0<=currentY &&
currentY<=image.getHeight()){
          currentY -= 1;
        }
        if(currentY<minY){
          minY = currentY;
        }
        lastMinY = currentY;
      } else {
        // I go up to the first black (or maxY)
        while(!(isSkinColor(getPixel(image, currentX,
currentY)))) {
          if(currentY > maxY +1){
            // There was no black pixels on this row
            moreToRight = false;
            maxX = currentX-1;
            break;
          }
          currentY += 1;
        }
        lastMinY = currentY;
```

```
      }

      if(moreToRight){
        // I go up to last black on this line
        while(isSkinColor(getPixel(image, currentX,
currentY))&& currentY<=image.getHeight()){
          currentY += 1;
        }
        if(currentY>maxY){
          maxY = currentY;
        }
        currentX += 1;
        currentY = minY;
      }
      currentY = lastMinY -1;
    }

    return new SkinColorArea(minX, maxX, minY, maxY);
  }

  public int getPixel(Image image, int x, int y){
    image.getRGB(rgbData, 0, 1, x, y, 1, 1);
    //System.out.println("x" + x + " y " + y);
    return rgbData[0];
  }

  public boolean isSkinColor(int pixelValue){

    //the pixelValue
    red   = (pixelValue & 0x00FF0000) >> 16;
    green = (pixelValue & 0x0000FF00) >> 8;
    blue  =  pixelValue & 0x000000FF;

    cb=(int) (0.148*red - 0.291*green + 0.439*blue +
128);
    cr=(int) (0.439*red - 0.368*green + 0.071*blue +
128);

    if (155<cb && cb<185){
      if(155<cr && cr<180){
        //System.out.println("true");
        return true;
      }else{
        return false;
      }
    }else{
      return false;
    }
  }
}
```

SkinColorArea.java

```
public class SkinColorArea {
  private int minX;
  private int maxX;
  private int minY;
  private int maxY;
  private int areaSize;

  public int centerX,centerY;

  public SkinColorArea(int minX, int maxX, int minY,
int maxY){
    this.minX = minX;
    this.maxX = maxX;
    this.minY = minY;
    this.maxY = maxY;
    calculateArea();
  }

  //calculates the area of the rectangular defined by
the 2 coodinates. This is to be able to compare it
  public void calculateArea(){
    this.areaSize = (this.maxX-this.minX)*(this.maxY-
this.minY);
  }

  public int getCenterX(){
    this.centerX=(int) ((this.minX + this.maxX) * 0.5);
    return this.centerX;
  }
```

```
  public int getCenterY(){
    this.centerY=(int) ((this.minY + this.maxY) * 0.5);
    return this.centerY;
  }

  public int getMaxX() {
    return maxX;
  }

  public void setMaxX(int maxX) {
    this.maxX = maxX;
  }

  public int getMaxY() {
    return maxY;
  }

  public void setMaxY(int maxY) {
    this.maxY = maxY;
  }

  public int getMinX() {
    return minX;
  }

  public void setMinX(int minX) {
    this.minX = minX;
  }

  public int getMinY() {
```

```
    return minY;
  }

  public void setMinY(int minY) {
    this.minY = minY;
  }

  public int getAreaSize() {
    return areaSize;
  }
```

```
  public String toString(){
//    "Skin colored area center: (" + getCenterX() +
"," + getCenterY() + ")";
    return "Skin colored area: (" + minX + "," + minY +
"), (" + maxX + "," + maxY + ")";
  }

}
```

Coordinate.java

```
/* Sets coordinate to the center of the biggest skin
area.
 * This will be either the face (when other skin parts
are shown
 * or the closest face when there are more than one
element on the picture
 */

import java.util.Vector;

public class Coordinate {
  private int x;
  private int y;

  public SkinColorArea skinColorA;

  //Constructor
  public Coordinate(int x, int y){
    this.x=x;
    this.y=y;
  }

  //Second constructor when a Vector of skinColorAreas
is sent
  public Coordinate (Vector skinColorAreas){
    int i = skinColorAreas.size();

    //it will give the center point of the only object
in the Vector.
    if(i==1){

  setCoordinateToCenterOfThisArea((SkinColorArea)skinC
olorAreas.elementAt(0));

    // if there are more than one element, it will
compare the sizes and choose the biggest.
    } else {
      skinColorA =
compareSkinColorAreas(skinColorAreas, i);
      setCoordinateToCenterOfThisArea(skinColorA);
    }
  }
```

```
  //compare the SkinColorAreas Vector to give the
biggest area back
  private SkinColorArea compareSkinColorAreas(Vector
skinColorAreas, int i) {
    int biggestElement=0;
    for(int j = 1; j<i;j++){
      if
(isSecondAreaBigger((SkinColorArea)skinColorAreas.elem
entAt(biggestElement),(SkinColorArea)skinColorAreas.el
ementAt(j))){
        biggestElement = j;
      }
    }
    return
(SkinColorArea)skinColorAreas.elementAt(biggestElement
);
  }

  //compares the area of two SkinColorAreas gives true
when second is bigger than first
  private boolean isSecondAreaBigger(SkinColorArea
area, SkinColorArea area2) {
    int areaS1 = area.getAreaSize();
    int areaS2 = area2.getAreaSize();
    if(areaS1<areaS2){
      return true;
    }else{
      return false;
    }
  }
  //sets coordinate to the center of the area sent to
this method
  private void
setCoordinateToCenterOfThisArea(SkinColorArea sc1) {
    this.x = (int) sc1.getCenterX();
    this.y = (int) sc1.getCenterY();
  }

  //to print out in the screen of the compiler
  public String toString(){
    return "(" + x + ", "+ y +")";
  }

}
```

Bibliography

1. **Hjelmås, Erik and Low, Boon Kee.** Face Detection: A Survey. *Computer Vision and Image Understanding.* 2001, Vol. 83, 3, pp. 236-274.

2. **Ming-Hsuan Yang, David J. Kriegman and Narendra Ahuja.** Detecting faces in images: A survey. 2002, pp. 34-58.

3. **Rein-Lien, Hsu; Abdel-Mottaleb, Mohamed; Anil, K. Jain.** Face Detection in Color Images. May 2002, Vol. 24, 5, pp. 696-706.

4. **Vezhnevets, Vladimir, Sazonov, Vassili and Andreeva, Alla.** *A Survey on Pixel-Based Skin Color Detection Techniques.* Moscow, Russia : s.n., 2003.

5. **Boehme, H.J.; Brakensiek, A.; Braumann, U.D.; Krabbes, M.; Gross, H.M.** Visually-Based Human-Machine-Interaction in a Nerural Architecture. *Supported by the Thuringian Ministry of Science, Research and Culture (TMWFK, GESTIK-Project).* 1997. http://citeseer.ist.psu.edu/466678.html.

6. **Turk, Matthew A.; Pentland, Alex P.** Eigenfaces for Recognition. *Journal of Cognitive Neuroscience.* 1991, pp. 586-591.

7. **Belhumeur, Peter N., Hespanha, João P. and Kriegman, David J.** Eigenfaces vs. Ficherfaces: Recognition Using Class Specific Linear Projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1997, Vol. 19, 7, pp. 711-720.

8. **Sciavicco, Lorenzo; Siciliano, Bruno.** *Modelling and Control of Robot Manipulators.* s.l. : Springer, 2000. 1852332212.

9. **Wikipedia.** [Online] 2007. [Cited: March 1, 2006.] http://en.wikipedia.org/wiki/.

10. **Merriam-Webster.** Merriam-Webster's Online Dictionary. [Online] Merriam-Webster, Incorporated, 2007. http://www.m-w.com/dictionary/.

11. *An Ethological and Emotional Basis for Human-Robot Interaction.* **Arkin, Ronald C.; Fujita, Masahiro; Takagi, Tsuyoshi; Hasegawa, Rika.** s.l. : Elsevier, 2003, pp. 191-201. 10.1016/S0921-8890(02)00375-5.

12. **Ling, Wong Hwee; Amin, Shamsudin H.M.** An Interactive Environment for a Mobile Robot Using Skin Detection. Skudai, Johor, Malasya : IEEE, 2003. 0-7803-8173.

13. **Marsic, Ivan; Medl, Attila; Flanagan, James.** Natural Communication With Information Systems. *Proceedings of the IEEE.* Aug 2000, Vol. 88, 8, pp. 1354-1366.

14. **Sobottka, Karin; Pitas, Ioannis.** Segmentation and Tracking of Faces in Color Images. 1996, pp. 236-241.

15. **Wang, Ce and Brandstein, Michael S.** A Hybrid Real-Time Face Tracking System. [ed.] ICASSP98. *ICASSP.* November 1997, pp. 1-4.

16. **Nagy, George; Zou, Jie.** Interactive Visual Pattern Recognition. 2002, Vol. 2, pp. 478-481.

17. *Computer Assisted Visual Interactive Recognition (CAVIAR) Technology.* **Evans, Arthur; Sikorski, John; Thomas, Patricia; Cha, Sung-Hyuk; Tappert, Charles.** NY : 2005 IEEE International Conference on Electro Information Technology, 2005.

18. **Chan, Y.H.; Abu-Bakar, S.A.R.** Face Detection System Based on Feature-Based Chromiance Colour Information. *Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (CGIV'04).* IEEE, 2004, 0-7695-2178.

19. **Yang, Guangzheng and Huang, T.S.** Human Face Detection in a Complex Background. [ed.] Pattern Recognition Society. *Pattern Recognition.* 1994, Vol. 27, 1, pp. 53-63.

20. **Singh, Sanjay Kr.; Chauhan, D.S.; Vatsa, Mayank; Singh, Richa.** A Robust Skin Color Based Face Detection Algorithm. India : s.n., 2003. Vol. 6, 4, pp. 227-234.

21. *Face Detectin by using Skin Color Model based on One Class Classifier.* **Hota, Rudra N., Venkoparao, Vijendran and Bedros, Saad.** Minneapolis, USA : 9th International Conference on Information Technology (ICIT'06) IEEE, 2006. 0-7695-2635-7/06.

22. **Rubino, Matthew T.** Edge Detection Algorithms. [Online] 07 2007. http://www.ccs.neu.edu/home/mtrubs/html/EdgeDetection.html.

23. **Rowley, Henry A.; Baluja, Shumeet; Kanade, Takeo.** Neural Network-Based Face Detection. *IEEE Transactions on Patterns Analysis and Machine Intelligence.* 1998, Vol. 20, 1.

24. **Bourke, Paul.** Autocorrelation -- 2D Pattern Identification. *Cross Correlation.* [Online] August 2007. http://astronomy.swin.edu.au/~pbourke/other/correlate/.

25. **Pentland, Alex, Moghaddam, Baback and Starner, Thad.** View-based and modular eigenspaces for face recognition. *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994.* 1994.

26. *Component-based Face Detection.* **Heisele, Bernd, et al.** s.l. : Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2001. CVPR 2001. , 2001.

27. *Automatic Feature Construction and a Simple Rule Induction Algorithm for Skin Detection.* **Gomez, Giovani and Morales, Eduardo F.** Mexico : Proc. of the ICML Workshop on Machine Learning in Computer Vision, 2002.

28. **Yanco, Holly A.; Drury, Jill L.; Scholtz, Jean.** Beyond Usability Evaluation: Analysis of Human-Robot. *Journal of Human-Computer Interaction.* 2004.

29. **Steinfeld, Aaron; Fong, Terrence; Kaber, David; Lewis, Michael; Scholtz, Jean; Schultz, Alan; Goodrich, Michael.** Common Metrics for Human-Robot Interaction. *HRI'06.* 2006.

30. **Burke, Jennifer L.; Murphy, Robin Roberson; Rogers, Erika; Lumelsky, Vladimir J.** Final Report for the DARPA/NSF Interdisciplinary Study on Human-Robot Interaction. *IEEE Transaction on Systems, Mand and Cybernetics.* 2, 2004, Vol. 34, Part C: Applications and Reviews.

31. **Koku, A.B.; Sekmen, A.; Alford, A.** Towards Socially Acceptable Robots. Nashville, TN 37235 : s.n., 2000. 0-7803-6583-6/2000 IEEE.

32. **Casper, Jennifer; Murphy, Robin Roberson;.** Human-Robot Interaction During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center. *IEEE Transaction on System, Man and Cybernetics, Part B.* June 2003, Vol. 33, 3.

33. *Workflow study on human-robot interaction in USAR.* **Casper, Jennifer; Murphy, R.;.** Washington, DC : IEEE, 2002. International Conference on Robotics & Automation. Vol. 2, pp. 1997-2003.

34. **Bruemmer, David J., et al.** Intelligent Robots for Use in Hazardous DOE Environments. *Proceedings of the Workshop on Measuring the Performance of Intelligent Systems.* August 2002.

35. **Hewett, Baecker,Card, Carey, Gasen, Mantei, Perlman, Strong and Verplank.** Curriculum Development Group of the Association for Computing Machinery, Special Interest Group on Computer-Human Interaction (ACM SIGCHI). [Online] 07 17, 2007. http://sigchi.org/cdg/cdg2.html#2_1.

36. **Scholtz, Jean C.** Human-Robot Interactions: Creating Synergistic Cyber Forces. *Multi-Robot Systems: From Swarms to Intelligent Automata (Proceedings from the 2002 conference).* 2002.

37. **Rogers, E., et al.** Cooperative Assistance for Remote Robot Supervision. *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on.* 1995, Vol. 5, pp. 4581-4586.

38. **Sun Developer Network.** J2ME Documentation Web site. *Diverse Data Sheets and Specifications of J2ME.* [Online] 2007. http://java.sun.com/j2me/docs/index.html.

39. **SunMicrosystems.** Sun Developer Network. *J2ME Building Blocks for Mobile Devices.* [Online] May 19, 2000. http://java.sun.com/products/cldc/wp/KVMwp.pdf.

40. **SonyEricsson.** Sony Ericsson Developer World. *White Paper Z520.* [Online] September 2005. http://developer.sonyericsson.com/.

41. **EclipseME.** J2ME Development using Eclipse. [Online] 2007. http://www.eclipseme.org/.

42. **Bray, Jennifer and Sturman, Charles F.** *Bluetooth: Connect Without Cables.* s.l. : Pretice Hall PTR, 2001. 0130898406.

43. **Bluetooth SIG.** *Specification of the Bluetooth System.* 2007.

44. **EZURiO.** Wireless M2M solutions. [Online] EZURiO, 2007. http://www.ezurio.com/products/industrial/.

45. Robotics and Intelligent Systems, research group. *University of Oslo.* [Online] 2007. http://www.ifi.uio.no/forskning/grupper/robin/.

46. **Høvin, Mats.** MES - ROBIN. [Online] 2007. http://heim.ifi.uio.no/~matsh/591188/index.html.

47. **Allegro Microsystems, Inc.** 2916 Dual full-bridge PWM Motor Driver. *Data Sheet.* 2916.

48. **Jack, Keith.** *Video Demystified: A Handbook for the Digital Engineer.* s.l. : Newnes, 2004. 0750678224.

49. **Foley, James D., et al.** *Computer Graphics: Principles and Practice.* s.l. : Addison-Wesley Publiching Company, Inc., 1990. 0-201-12110-7.

50. *An efficient architecture for color space conversion using Distributed Arithmetic.* **Bensaali, F., Amira, A. and Bouridane, A.** Belfast, UK : Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. ISCAS '04. , 2004. 0-7803-8251-X.

51. *A Novel Skin Color Model in YCbCr Color Space and its Application to Human Face Detection.* **Son Lam Phung, Abdesselam Bouzerdoum, Douglas Chai.** Australia : International Conference on Image Processing, 2002. 0-7803-7622-06.

52. **Borenstein, Johann; Ulrich, Iwan.** The Guidecane - A Computerized Travel Aid. Albuquerque, New Mexico : Proceedings of the 1997 IEEE Intemational Conference on Robotics and Automation, 1997.

53. **Jung-Hoon Hwang, Kang Woo Lee, and Dong-Soo Kwon.** A Formal Model of Sharing Grounds for Human-Robot Interaction. *The 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN06).* September 6-8, 2006, pp. 298-303.

54. **Berthold Klaus, Paul Horn.** *Robot Vision.* s.l. : MIT Press, 1986. 0262081598.

55. **Nitzan, David.** Development of Intelligent Robots: Achievements and Issues. *IEEE JOURNAL OF ROBOTICS AND AUTOMATION.* 1985, Vols. RA-1, 1.

56. *Detecting and Tracking Eyes By Using Their Physiological Properties, Dynamics and Appearance.* **Haro, Antonio; Flickner, Myron; Essa, Irfan.** 1063-6919/00, 2000, Procceedings IEEE CVPR, pp. 163,168.

57. *Assessing Human Likenes by Eye Contact in an Android Testbed.* **MacDorman, Karl F.; Minato, Takashi; Shimada, Michihiro; Itakura, Shoji; Cowley, Stephen; Ishiguro, Hiroshi.** Kyoto, Japan : CogSci2005- XXVII Annual Conference of the Cognitive Science Society, 2005.

58. **Lee, Taigun; Park, Sung-Kee; Park, Mignon.** A New Facial Features and Face Detection Method for Human-Robot Interaction. *Proceedings of the 2005 IEEE.* International Conference on Robotics and Automation, April 2005.

59. **Ortiz, C. Enfique; Giguère, Eric.** *Mobile information device profile for Java 2 Micro Edition : professional developer's guide.* New York : Wiley, 2001.

60. **Venetsanoupoulos, Anastasios N. and Plataniotis, Konstantinos N.** *Color Image Processing and Applications.* s.l. : Springer, 2000. 3540669531.

61. Eye contact. *Cambridge Advanced Learner's Dictionary.* May 14, 2006.

62. **Simion, Franesca; Cassia, Viola Macchi; Turati, Chiara; Valenza, Eloisa.** The Origins of Face Perception: Specific Versus Non-specific Mechanism. [ed.] Ltd John Willey & Sons. *Infant and Child Development.* 10, 2001, Vol. 10.1002/icd.247, pp. 59-65.

63. **Li, Stan Z.; Jain, Anil K.** *Handbook of Face Recognition.* New York : Springer, 2005. 0-387-40595-X.

64. **Bagci, A.M.; Ansari, R.; Khokhar, A.; Cetin, E.** Eye Tracking Using Markov Models. 2004.

65. **Thayer, S.** Children's detection of on-face and of-face gazes. 1977, Vol. 13, 673-674.

66. *Human Face Detection in Cluttered Color Images Using Skin Color and Edge Information.* **Sandeep, K.; Rajagopalan, A.N.** India : s.n., 2005.